Title: LASSI Software Architecture Document	Author: N. Sizemore	Date: August 30, 2019
GBO Doc. NRAO-391-741		Version: 0.6

Laser Antenna Surface Scanning Instrument (LASSI) Software Architecture Document (SAD)

PREPARED BY	**ORGANIZATION**
Nathaniel D. Sizemore (nsizemor@nrao.edu)	Green Bank Observatory

Change Record

VERSION	DATE	REASON
0.1		Initial draft
0.2	2019-06-19	Refinement after several ADD iterations
0.3	2019-07-12	Changes to clarify goals vs. requirements of project
0.4	2019-07-22	Changes after review by project management
0.5	2019-08-07	Agreement with other project documents
0.6	2019-08-30	Changes in response to PDR RIDs
0.7	2019-09-03	Updates for RID responses

BACKGROUND

This template is based on the Software Engineering Institute's "View and Beyond" method for documenting software architectures, as described in Clements, et al., *Documenting Software Architecture: Views and Beyond* (Addison Wesley, 2002). The current version is available for free download from the SEI's architecture web site.

TIPS FOR USING THIS TEMPLATE

To create an instance of this document:

- Insert relevant information on cover sheet and in placeholders throughout.
- Insert relevant information in page header: Move to a page of the body of the report, select *View > Header and Footer* from the main menu, and then replace relevant information in the header box at the top of the page.

To update the contents and page numbers in the Table of Contents, List of Figures, and List of Tables:

- Position the cursor anywhere in the table to be updated.
- Click the *F9* function key.
- Answer "Update entire table".

To insert a figure or table caption:

- From the main menu, choose *Insert > Reference > Caption* and then either *Figure* or *Table* as needed.
- Click the OK button.
- Add a colon and a tab stop after the figure number in the caption itself.
- The caption should use the *Caption* style.
- Add a colon and a tab stop after the table/figure number in the caption itself.

TIPS FOR MAKING YOUR DOCUMENT MORE READABLE

- A gray box containing *CONTENTS OF THIS SECTION* is provided at the beginning of most sections and subsections. After determining what specific information will be included in your document, you can remove this gray box or leave it to serve as a quick-reference section overview for your readers. In the case that text has been provided in the template, inspect it for relevance and revised as necessary.
- Consider hyperlinking key words used in the document with their entries in the Glossary or other location in which they are defined. Choose *Insert > Hyperlink*.
- Don't leave blank sections in the document. Mark them "To be determined" (ideally with a promise of a date or release number by which the information will be provided) or "Not applicable."
- Consider packaging your SAD as a multi-volume set of documentation. It is often helpful to break your documentation into more than one volume so that the document does not become unwieldy. There are many ways that this can be accomplished. The structuring of the document must support the needs of the intended audience and must be determined in the context of the project. Each document that you produce should include the date of issue and status; draft, baseline, version number, name of issuing organization; change history; and a summary. A few decomposition options are:
    - *A 2-Volume approach:* Separate the documentation into two volumes; one that contains the views of the software architecture and one that contains everything else. A common variant of this approach has one volume per view, and one volume for everything else.
    - *A 3-Volume approach:* Document organizational policies, procedures, and the directory in one volume, system specific overview material in a second, and view documentation in a third.
    - *A 4-Volume approach:* Create one volume for each viewtype [Clements 2002] (module, component-and-connector, allocation) that contains the documentation for the relevant views. Include all of the other information in the fourth volume.
    - Software interfaces are often documented in a separate volume.

  In *any* case, the information should be arranged so that readers begin with the volume containing the Documentation Roadmap (Section 1 in this template).

Table of Contents

1 Documentation

This document describes the LASSI software architecture, and is organized as follows:

- Section 1.1 explains revision history. This tells you if you're looking at the correct version of the SAD.
- Section 1.2 explains the purpose and scope of the SAD, and indicates what information is and is not included. This tells you if the information you're seeking is likely to be in this document.
- Section 1.3 explains the information that is found in each section of the SAD. This tells you what section(s) in this SAD are most likely to contain the information you seek.
- Section 1.4 explains the stakeholders for which the SAD has been particularly aimed. This tells you how you might use the SAD to do your job.
- Section 1.5 explains the standard organization used to document architectural views in this SAD. This tells you what section within a view you should read in order to find the information you seek.

1.1 Document Management and Configuration Control Information

- Revision Number: 0.7
- Revision Release Date: 2019-09-03
- Purpose of Revision: Updates for RID responses
- Scope of Revision: Appendix: Design History

1.2 Purpose and Scope of the SAD

CONTENTS OF THIS SECTION: This section explains the SAD's overall purpose and scope, the criteria for deciding which design decisions are architectural (and therefore documented in the SAD), and which design decisions are non-architectural (and therefore documented elsewhere).

This SAD specifies the software architecture for the LASSI project**.** All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents.

What is software architecture? The software architecture for a system[1] is the structure or structures of that system, which comprise software elements, the externally-visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties

[1] Here, a system may refer to a system of systems.

refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD, and what information is relegated to downstream documentation.

Elements and relationships. The software architecture first and foremost embodies information about how the elements relate to each other. This means that architecture specifically omits certain information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

Multiple structures. The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and podiatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement's behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So it is with software. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

For example, all non-trivial software systems are partitioned into implementation units; these units are given specific responsibilities, and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to sub-teams. This is one kind of structure often used to describe a system. It is a very static structure, in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system's function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units described previously that are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

None of these structures alone is *the* architecture, although they all convey architectural information. The architecture consists of these structures as well as many others. This example shows that since architecture can comprise more than one kind of structure, there is more than one kind of element (e.g., implementation unit and processes), more than one kind of interaction among elements (e.g., subdivision and synchronization), and even more than one context (e.g., development time versus runtime). By intention, the definition does not specify what the architectural elements and relationships are. Is a software element an object? A process? A library? A database? A commercial product? It can be any of these things and more.

These structures will be represented in the views of the software architecture that are provided in Section 3.

Behavior. Although software architecture tends to focus on structural information, *behavior of each element is part of the software architecture* insofar as that behavior can be observed or discerned from the point of view of another element. This behavior is what allows elements to interact with each other, which is clearly part of the software architecture and will be documented in the SAD as such. Behavior is documented in the element catalog of each view.

1.3 How the SAD Is Organized

CONTENTS OF THIS SECTION: This section provides a narrative description of the major sections of the SAD and the overall contents of each.   Readers seeking specific information can use this section to help them locate it more quickly.

This SAD is organized into the following sections:

- **Section 1 provides information about this document and its intended audience**. It provides the roadmap and document overview.   Every reader who wishes to find information relevant to the software architecture described in this document should begin by reading Section 1, which describes how the document is organized, which stakeholder viewpoints are represented, how stakeholders are expected to use it, and where information may be found. Section 1 also provides information about the views that are used by this SAD to communicate the software architecture.
- **Section 2 explains why the architecture is what it is.** It provides a system overview, establishing the context and goals for the development. It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture, and it describes the major architectural approaches that have been utilized in the architecture. It includes information about evaluation or validation performed on the architecture to provide assurance it meets its goals.
- **Section 3 and Section 4 specify the software architecture**. Views specify elements of software and the relationships between them. A view is a representation of one or more structures present in the software (see Section 1.2).

- **Sections 5 and 6 provide reference information for the reader.** Section 5 provides look-up information for documents that are cited elsewhere in this SAD. Section 6 is a *directory*, which is an index of architectural elements and relations telling where each one is defined and used in this SAD. The section also includes a glossary and acronym list.

1.4 Stakeholder Representation

This section provides a list of the stakeholder roles considered in the development of the architecture described by this SAD. For each, the section lists the concerns that the stakeholder has that can be addressed by the information in this SAD.

Each stakeholder of a software system—customer, user, project manager, coder, analyst, tester, and so on—is concerned with different characteristics of the system that are affected by its software architecture. For example, the user is concerned that the system is reliable and available when needed; the customer is concerned that the architecture can be implemented on schedule and to budget; the manager is worried (in addition to cost and schedule) that the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways. The developer is worried about strategies to achieve all of those goals. The security analyst is concerned that the system will meet its information assurance requirements, and the performance analyst is similarly concerned with it satisfying real-time deadlines.

This information is represented as a matrix, where the rows list stakeholder roles, the columns list concerns, and a cell in the matrix contains an indication of how serious the concern is to a stakeholder in that role.

	Accuracy	Performance	Interoperability	Usability	Resource Mgt	Modifiablity
Scientists	X	X				
Mechanical Engineering			X			
Electrical Engineering			X			
Software Development			X			X
Project Manager		X			X	
Observer (end users)	X	X		X		
Operations				X		
Safety				X		
GBO Management					X	

1.5 How a View is Documented

Each view is documented as follows. Some sections may be left out if they do not apply to a given view. Unless otherwise specified, diagrams are presented using UML 2.0.

- Name of view.
- Primary presentation. This section presents the elements and the relations among them that populate this view, using an appropriate language, languages, notation, or tool-based representation.
- Element catalog. Whereas the primary presentation shows the important elements and relations of the view, this section provides additional information needed to complete the architectural picture. It consists of the following subsections:
- Elements. This section describes each element shown in the primary presentation, details its responsibilities of each element, and specifies values of the elements' relevant *properties*, which are defined in the viewpoint to which this view conforms.
- Relations. This section describes any additional relations among elements shown in the primary presentation, or specializations or restrictions on the relations shown in the primary presentation.
- Interfaces. This section specifies the software interfaces to any elements shown in the primary presentation that must be visible to other elements.
- Behavior. This section specifies any significant behavior of elements or groups of interacting elements shown in the primary presentation.
- Constraints: This section lists any constraints on elements or relations not otherwise described.
- Context diagram. This section provides a context diagram showing the context of the part of the system represented by this view. It also designates the view's scope with a distinguished symbol, and shows interactions with external entities in the vocabulary of the view.
- Variability mechanisms. This section describes any variability that are available in the portion of the system shown in the view, along with how and when those mechanisms may be exercised.
- Architecture background. This section provides rationale for any significant design decisions whose scope is limited to this view.
- Relation to other views. This section provides references for related views, including the parent, children, and siblings of this views.

NOTE: The LASSI software architecture is updated as prototyping and other research and development tasks are completed. Elements of the SAD to be determined based on ongoing research and development work are marked "TBD". These elements will be refined and elaborated as the project moves moves from preliminary to detailed design.

1.6 Process for Updating this SAD

If you find omissions, errors, ambiguities, or other content that needs to be updated, please contact the document author, Nathaniel D. Sizemore, at nsizemor@nrao.edu. The most recent published version of this document will be available on the LASSI SharePoint site.

2 Architecture Background

2.1 Problem Background

2.1.1 System Overview

The goal of the LASSI project is to explore the feasibility of expanding the high-frequency observing time of the GBT by use of a commercial off-the-shelf (COTS) laser metrology scanner. Using this device, we will:

1. calculate the current shape of the GBT primary reflector,
2. compare it to a desired reference shape, and
3. issue commands to the GBT monitor and control (M&C) system to change the active surface back to the desired reference surface.

The LASSI software architecture is designed to control the scanner, implement a data pipeline for needed data reduction and transformation, and interface with other GBT systems.

More details on the overall goals for LASSI are stated in the GBT Metrology ConOps document, available on SharePoint.

2.1.2 Goals and Context

CONTENTS OF THIS SECTION: This section describes the goals and major contextual factors for the software architecture. The section includes a description of the role software architecture plays in the life cycle, the relationship to system engineering results and artifacts, and any other relevant factors.

The software architecture for LASSI describes a system enabling users to obtain data from the TLS scanner, use that data to calculate the surface of the primary reflector, compare to a reference surface, and issue any corrections. All of these steps must be performed in wallclock time on par with existing AutoOOF corrections. (For details on the wallclock goals, see section 2.1.3.)

LASSI includes both hardware installed on the GBT structure and software that is part of the GBT's M&C system. This environment places various constraints on LASSI, including the need to adhere to RFI limits and using existing M&C communication protocols.

For the complete context and goals, see the ConOps document, referenced in section 2.1.

2.1.3 Significant Driving Requirements

CONTENTS OF THIS SECTION: This section describes behavioral and quality attribute requirements (original or derived) that shaped the software architecture. Included are any scenarios that express driving behavioral and quality attribute goals, such as those crafted during a Quality Attribute Workshop (QAW) [Barbacci 2003] or software architecture evaluation using the Architecture Tradeoff Analysis Method^{SM} (ATAM^{SM}) [Bass 2003].

The ConOps and Architecture Plan documents describe the envisioned plan of operation, as well as seed scenarios and use cases for LASSI.

Primary functional requirements include:

- A system capable of investigating the feasibility of using the TLS scanner to reduce the time needed to calibrate the GBT for high-frequency observations,
- A system capable of investigating the feasibility of detecting differences in the primary reflector based on consecutive measurements.

Quality attributes that have been defined include:

- Accuracy – LASSI must be able to calculate a surface that correctly models the primary reflector of the GBT to within the noise budget.
- Performance – Under normal conditions, the LASSI procedure should take less than 25 minutes of wallclock time – beginning when the hardware is positioned correctly and a user requests a LASSI measurement, and ending when active surface changes are calculated and ready to be issued to the M&C system. This target will allow LASSI to match AutoOOF's optics quality within the same amount of time.
- Availability – TLS measurement data and any needed corrections must be accessible during observations. Note that high-frequency observations and use of the TLS scanner hardware are constrained by weather.
- Reliability – GBO support staff must be able to abort a TLS scan during normal operations, bringing the M&C system to a state ready to accept the next command within 60 seconds.

Constraints that this architecture must abide by include:

- Leica software interface is provided only through Microsoft .NET libraries.
- Physical mounting limitations for placing the TLS scanner on the GBT structure.
- RFI constraints due to location at GBO; the TLS scanner should be powered off when not in use.

^{S} ADD^{tm} Quality Attribute Workshop and QAW and Architecture Tradeoff Analysis Method and ATAM are service marks of Carnegie Mellon University.

- LASSI must respect bandwidth constraints and be a “good citizen” on the GBT internal network. Given the amount of data and the current 100 Mbit connection to be used, GBO’s IT staff does not anticipate any issues.

Architectural concerns this design addresses include:

- System integration with GBT M&C (Interoperability)
- Use of software best practices (Modifibility)

2.2 Solution Background

CONTENTS OF THIS SECTION: The sub-parts of Section 2.2 provide a description of why the architecture is the way that it is, and a convincing argument that the architecture is the right one to satisfy the behavioral and quality attribute goals levied upon it.

2.2.1 Architectural Approaches

CONTENTS OF THIS SECTION: This section provides a rationale for the major design decisions embodied by the software architecture. It describes any design approaches applied to the software architecture, including the use of architectural styles or design patterns, when the scope of those approaches transcends any single architectural view. The section also provides a rationale for the selection of those approaches. It also describes any significant alternatives that were seriously considered and why they were ultimately rejected. The section describes any relevant COTS issues, including any associated trade studies.

LASSI requires that data flow from the scanner hardware, through a series of manipulations and reductions, and end in commands to the GBT’s active surface. Since these steps must be done in order, this workflow maps to the pipe-and-filter architecture pattern. This decision is supported by the characteristics of pipe-and-filter as described in Bass 2003:

> *Data transformation systems are typically structured as pipes and filters . . . Often such systems constitute the front end of signal-processing applications. These systems receive sensor data at a set of initial filters; each of these filters compresses the data and performs initial processing (such as smoothing). Downstream filters reduce the data further and do synthesis across data derived from different sensors. The final filter typically passes its data to an application, for example providing input to modeling or visualization tools.*

The description of this architecture pattern closely matches the desired behavior for LASSI, and is used as the overall reference architecture. However, due to the requirement to integrate with the existing GBT M&C system, the LASSI architecture has been modified from pipe-and-filter to include functionality for monitoring, control, and coordination.

2.2.2 Analysis Results

CONTENTS OF THIS SECTION: This section describes the results of any quantitative or qualitative analyses that have been performed that provide evidence that the software architecture is fit for purpose. If an Architecture Tradeoff Analysis Method evaluation has been performed, it is included in the analysis sections of its final report. This section refers to the results of any other relevant trade studies, quantitative modeling, or other analysis results.

Other reference architectures were considered, including client-server, peer-to-peer, and a layered approach. However, the GBO software engineering group determined that none of these alternatives captured the linear nature of LASSI, given both how the system was envisioned in the ConOps document and the nature of similar signal-processing procedures performed by the GBT.

To ensure that the overall approach of using COTS hardware was viable, and that the resulting data could be used as desired, multiple explorations were performed: detailed information on these investigations can be found in the following documents.

- March 2016: https://www.cv.nrao.edu/~fschwab/consub2.ex1.html
- June 2019: https://sharepoint.nrao.edu/pmd/projects/_layouts/15/WopiFrame.aspx?sourcedoc=/pmd/projects/613%20GBT%20Metrology/Software/Data%20Analysis%20Pipeline/PDFs/6jun2019/Scans%209%20vs%2011%20-%20Z4%20detected%20.pdf&action=default

2.2.3 Requirements Coverage

CONTENTS OF THIS SECTION: This section describes the requirements (original or derived) addressed by the software architecture, with a short statement about where in the architecture each requirement is addressed.

The following table enumerates the use cases described in the GBT Metrology Architecture Plan, the related quality attributes being addressed, and references the views detailing how the requirement is being satisfied.

ID	Quality Attribute	Associated Use case	Related View
QA-1	Usability	Execute AutoOOF - GBO observers use the current out-of-focus holography procedure (AutoOOF) to reduce surface RMS for the GBT antenna. Corrections are encoded by Zernike polynomials to express surface corrections for the Antenna Control System (ACS) to adjust the antenna shape (see RD-3).	3.1 of Architecture Plan
QA-2	Usability	Conduct initial TLS measurement - Immediately after an AutoOOF measurement, GBO observers use an AstrID script to cause the Leica scanner	3.2, 3.8

ID	Quality Attribute	Associated Use case	Related View
		to measure the dish, send the resulting data to a data processor, and store the results.	
QA-3	Testability	Store TLS results - During an initial TLS scan (4.2), the scan results (both point cloud and reducted data) are stored in /home/gbtdata.	3.8
QA-4	Interoperability	Start Observation - GBO observers use the existing AstrID interface to start a TLS measurement.	3.1, 3.8
QA-5	Accuracy	Conduct TLS Measurement - Sometime after the initial TLS measurement, GBO observers use an AstrID script to execute and compare the new results with the previous results and generate antenna corrections which are sent to the ACS.	3.2, 3.8
QA-6	Accuracy	Compare TLS Results - An algorithm (currently under development) compares previous measurement data to current measurement data and calculates new corrections for consumption by the ACS.	3.7
QA-7	Interoperability	Change the Shape of the GBT Antenna – LASSI causes corrections generated to be sent to the ACS.	3.2
QA-8	Accuracy	Verify Results - GBO observers use the AstrID Peak and Focus procedures to verify that the TLS-based corrections adjusted the shape of the antenna as expected. This use case will primarily occur during LASSI development and testing.	GBO Observers' Guide
QA-9	Usability	User confirmation – before changes are issued, users will be presented with the proposed changes calculated by LASSI, and will be able to accept or reject those changes.	TBD
QA-10	Accuracy	Telescope azimuth and elevation position shall be captured at the beginning and end of a measurement. This will allow a flag to be set if there is a difference.	TBD

3 Views

This section contains the views of the software architecture. A view is a representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. Concretely, a view shows a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.

Architectural views can be divided into three groups, depending on the broad nature of the elements they show. These are:

- Module views. Here, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned areas of functional responsibility, and are assigned to teams for implementation. There is less emphasis on how the resulting software manifests itself at runtime. Module structures allow us to answer questions such as: What is the primary functional responsibility assigned to each module? What other software elements is a module allowed to use? What other software does it actually use? What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?
- Component-and-connector views. Here, the elements are runtime components (which are principal units of computation) and connectors (which are the communication vehicles among components). Component and connector structures help answer questions such as: What are the major executing components and how do they interact? What are the major shared data stores? Which parts of the system are replicated? How does data progress through the system? What parts of the system can run in parallel? How can the system's structure change as it executes?
- Allocation views. These views show the relationship between the software elements and elements in one or more external environments in which the software is created and executed. Allocation structures answer questions such as: What processor does each software element execute on? In what files is each element stored during development, testing, and system building? What is the assignment of the software element to development teams?

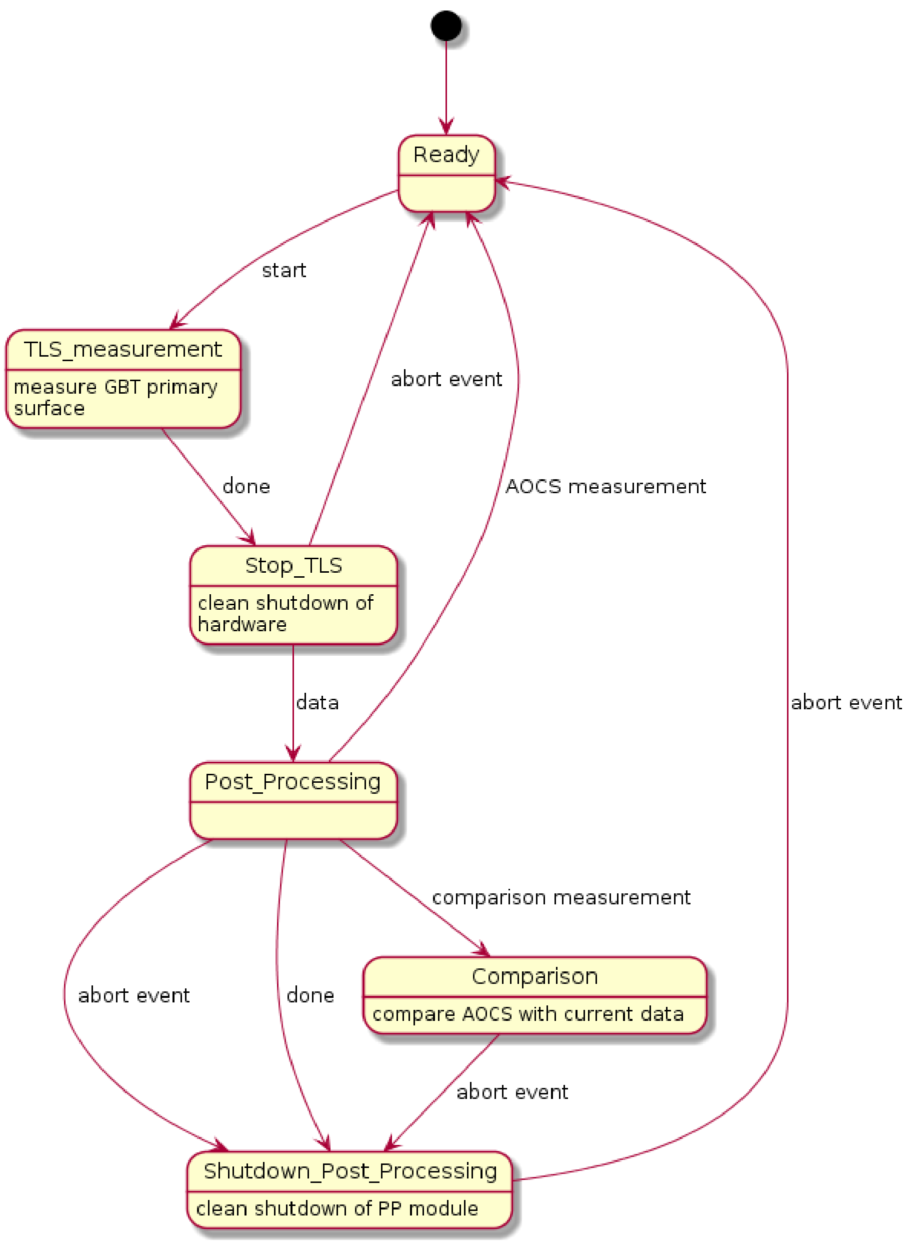
These three kinds of structures correspond to the three broad kinds of decisions that architectural design involves:

- How is the system to be structured as a set of code units (modules)
- How is the system to be structured as a set of elements that have run-time behavior (components) and interactions (connectors)?
- How is the system to relate to non-software structures in its environment (such as CPUs, file systems, networks, development teams, etc.)?

Often, a view shows information from more than one of these categories. However, unless chosen carefully, the information in such a hybrid view can be confusing and not well understood.

3.1 LASSI State View

3.1.1 Primary Presentation

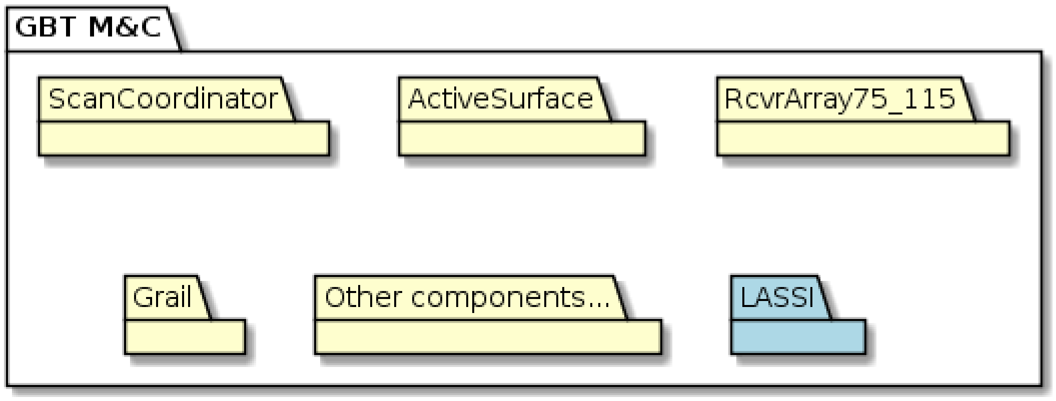


3.1.2 Element Catalog

- Ready – the default state for LASSI

- TLS_measurement – LASSI using the DAQ module to conduct a TLS measurement of the GBT primary surface
- Post_Processing – LASSI running the post-processing module, reducing the raw data from the TLS scanner and calculating Zernike polynomials for the surface. The measurement directly following the correction of the primary surface by AutoOOF is referred to as an “AutoOOF Corrected Surface”, or AOCS.
- Stop_TLS – the system will cleanly stop the scanner and perform any associated tasks.
- Comparison – the post-processing module will compare the results of two TLS measurements, calculating the difference between them in the form of a Zernike polynomial.
- Shutdown_Post_Processing – perform any needed steps to cleanly stop the post-processing module.

3.1.3 Context Diagram



3.1.4 Architecture Background

LASSI must go through an ordered series of steps to measure the GBT prime reflector, calculate a surface, and issue any needed corrections. However, as it is integrated into the the larger GBT M&C system, it must correctly respond to events while the data through its pipeline. The two anticipated events that might happen during LASSI’s execution are:

1. The user aborts the procedure.
2. An error or failure is detected at any point in the pipeline.

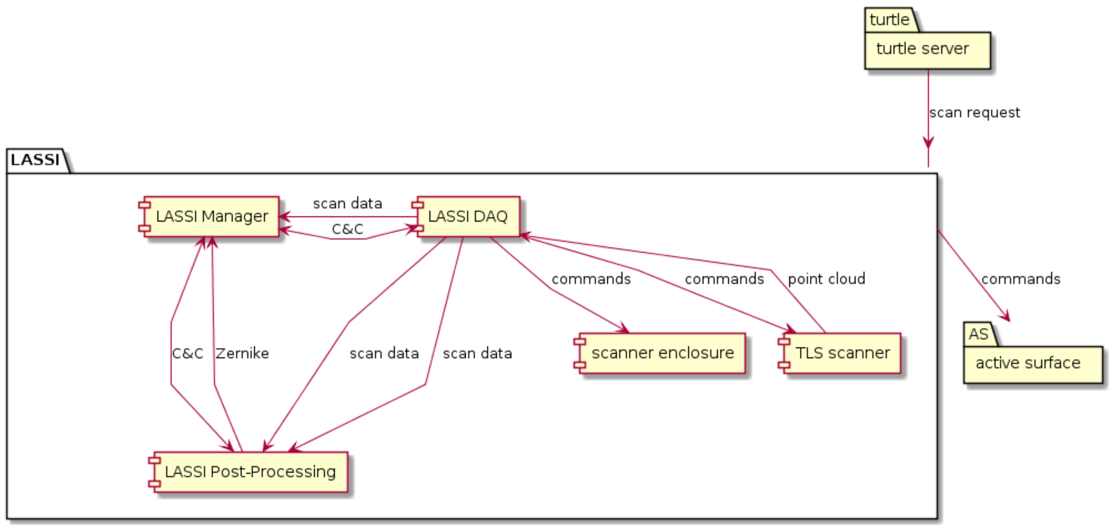
Both events are handled in the same way, with LASSI providing an exit path at each step in the pipe-and-filter architecture to cleanly stop operations at any point in the process.

3.1.5 Related Views

TBD – This section will be developed during the detailed design phase.

3.2 Component-and-Connector View

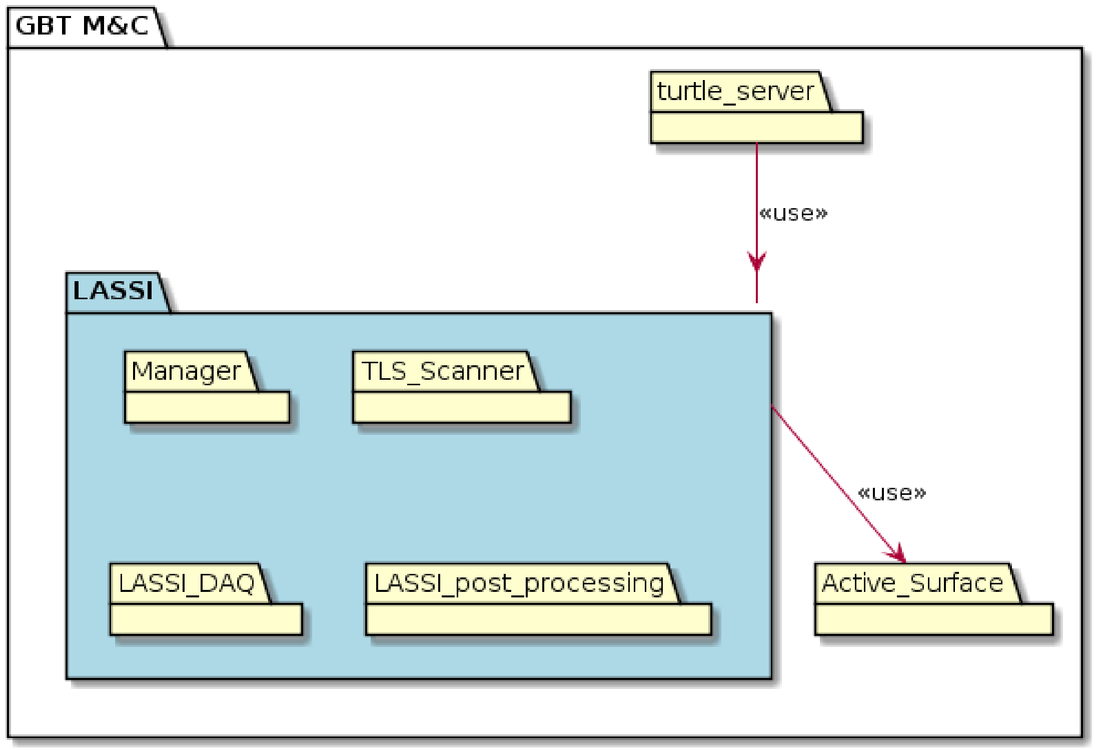
3.2.1 Primary Presentation



3.2.2 Element Catalog

- Turtle server – user-facing software that issues commands to the telescope; AstrID is a GUI Turtle client.
- TLS Scanner – Leica scanner hardware.
- scanner enclosure – environmental housing for the TLS Scanner, and any other associated mechanical components required for operation on the GBT structure.
- LASSI DAQ – software commanding the TLS scanner and transferring data to and from it. The output of this component will be data from the TLS measurement in (x, y, z, I) format.
- LASSI manager – software used to coordinate, monitor, and control the LASSI system.
- LASSI post-processing – software that takes in point clouds from the scanner, does any required data reduction, and returns calculated Zernike coefficients that describe the GBT active surface.
- GBT M&C – GBT Monitor and Control system.

3.2.3 Context Diagram



3.2.4 Architecture Background

This structure is based on the minimum system prototype (MSP) described in the ConOps documentation, Appendix A. Note that as such, it does *not* include features such as storing the surface data as shown in Figure 3 of the ConOps.

Commanding the initial AutoOOF corrections that LASSI attempts to conform to on subsequent runs is handled by AstrID, and is not part of the LASSI architecture. Similarly, while LASSI will calculate the needed Zernike polynomials to make adjustments to the shape of the prime focus, the actual corrections to the dish are made by the GBT M&C system.

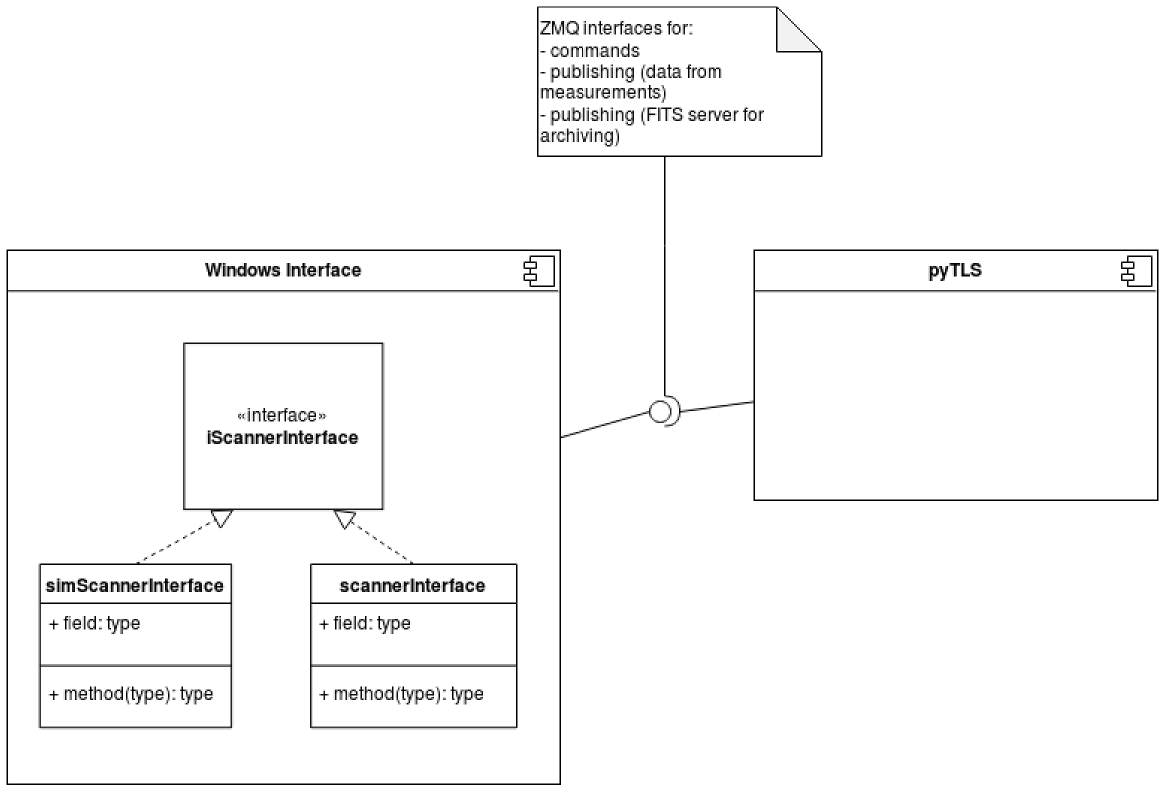
The TLS scanner will be mounted to ensure as much of the resulting data is of the primary surface, minimizing views of the ground, superstructure, etc. as much as possible.

3.2.5 Related Views

TBD – This section will be developed during the detailed design phase.

3.3 LASSI DAQ Decomposition View

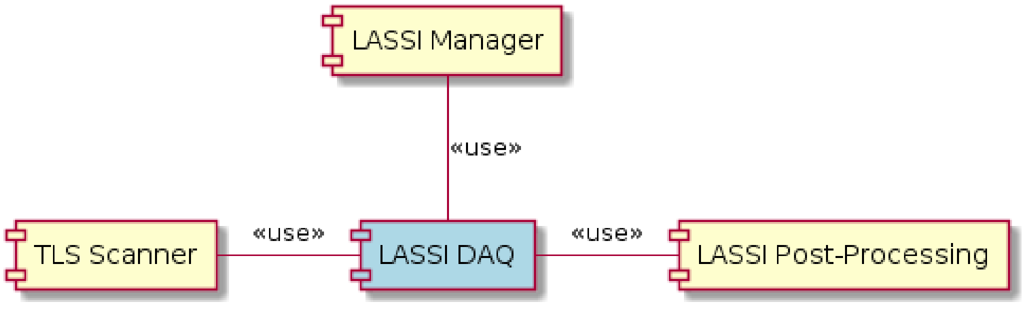
3.3.1 Primary Presentation



3.3.2 Element Catalog

- Windows Interface – component running on Windows, used for command and control of the TLS scanner.
- PyTLS – a Python interface to the TLS scanner, allowing platform-independent communication and control of the device.

3.3.3 Context Diagram



3.3.4 Architecture Background

The LASSI DAQ is constrained by the Windows software libraries provided by Leica to command and control the TLS scanner remotely. As such, the LASSI DAQ needed to be designed in such a a way that the Windows software could be used to control the hardware, while at the same time providing an interface for the rest of LASSI and the GBT M&C system, all running on Linux.

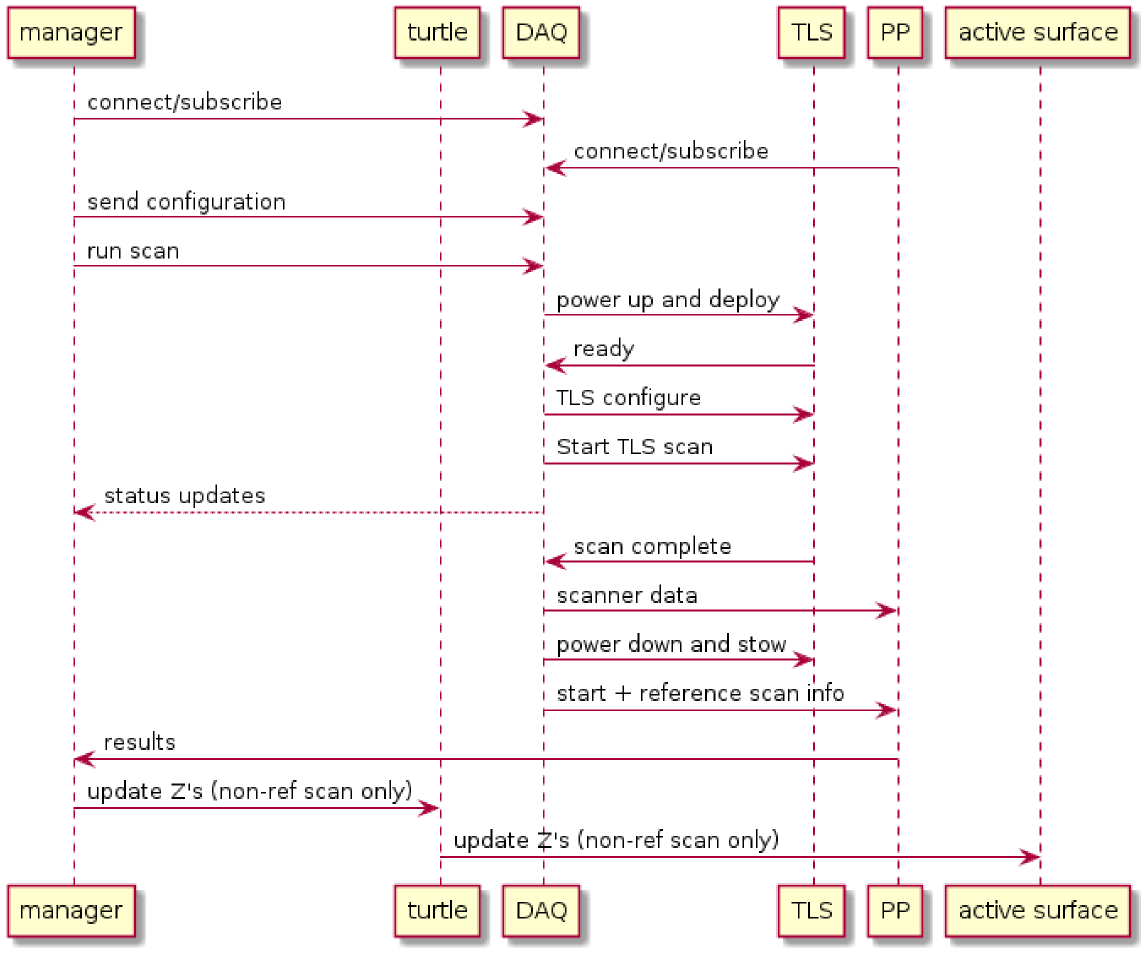
We are investigating whether the TLS scanner libraries used by the LASSI DAQ can take advantage of GPU-enabled hosts to speed up data reduction.

3.3.5 Related Views

- Deployment View
- https://sharepoint.nrao.edu/pmd/projects/613%20GBT%20Metrology/Software/Architecture%20Plan/LASSI_DAQ_SAD.odt?Web=1

3.4 LASSI DAQ Sequence View

3.4.1 Primary Presentation



3.4.2 Element Catalog

- Manager – the LASSI manager; the software abstraction used by the GBT M&C system for monitor and control of LASSI.
- DAQ – software commanding the TLS scanner and transferring data to and from it. The output of this component will be data from the TLS measurement in (x, y, z, I) format.
- TLS – Leica scanner hardware.

- PP – GBO-developed software that takes in point clouds from the scanner, does any required data reduction, and returns calculated Zernike coefficients that describe the GBT active surface.
- Turtle – the observation management system that loads, edits, and validates Scheduling Blocks, submits them to the telescope for execution immediately or dynamically, and monitors the progress of execution.
- active surface – the software controlling the active surface of the GBT that can adjust the shape of the dish using Zernike polynomials as input.

The communication between various participants are described below.

TLS Power up and deploy

Via a remote power switch or power strip (e.g. 'iboot bar'), power is applied to the unit. It should be noted that once shutdown, the unit should not be re-powered on for a TBD interval, to allow residual voltages to bleed off. (We may need an interval timer to prevent short power-off periods.) The unit powers up and boots up. This operation also opens the weather enclosure via the same power up signal.

TLS Network Connection

In order for the unit to connect onto a network, it must first be assigned an IP address via DHCP. Once assigned an address, the unit listens for broadcast messages from the LASSI DAQ program. A connection is negotiated by responding to the broadcasts with an 'I am here' message. The lassi_daq program will monitor and try to initiate connection after power is applied to the TLS.

Setup

The LASSI DAQ program monitors the status of the instrument. Once the LASSI DAQ detects a status of ready, it will specify the measurement parameters, such as:

- range vs. speed (select range)
- distance (100m)
- scan field of view (FOV)
- number of image points
- TLS project name
- weather temp, press, and atmospheric scale (PPM)

Note that the "project name" in this context is part of the metadata that the TLS device tracks and is used for managing TLS measurement data on the the device itself (such as when it writes point clouds to its own local disk). It should not be confused with a GBT observation project identifier. This metadata is useful for engineering and development, and we will explore using the GBT project name as the TLS project name as we move into the detailed design phase.

Execute Scan

The execute scan event is sent to the device. The device executes whatever calibrations are necessary and then begins the scan. The LASSI DAQ monitors the scanner status to monitor scan progression. We are currently investigating the best method of synchronizing the scanner time with the rest of the system.

Scan Data Transfer/Export

Once the scan is complete, calls are made to the TLS API to extract the TLS scan data, and then publishes the scan data to both the LASSI manager (which write a FITS file for archival purposes) and the post-processing programs.

Scanner Power Down

Once the scan is complete, and the data has been transferred, the LASSI DAQ will issue a power-down event to the scanner, and the TLS performs an orderly shutdown. A timer will delay for a period and then remove power from the TLS unit. This will also close the weather housing for the scanner hardware.

LASSI Interfaces

The primary interface used by both the LASSI manager and the post-processing programs is implemented using ZeroMQ (pronounced ‘zero-M-Q’) communications library, and the Mesgpack data encodings. The communications pattern will be via PUSH/PULL pair for commands and status requests, and PUB-SUB for asynchronous results and data.

TLS Power Control

As an example, the ‘iboot’ remote control power strip is commanded using an HTTP protocol. The actual unit to be used for this is TBD – this section will be developed during the detailed design phase.

Specify and Perform Scan

The scan parameters will be specified by the LASSI manager. The scan parameters and start message can be sent without waiting for the TLS deployment/power-up. The LASSI DAQ will handle the sequencing of the required steps.

Scan Parameters:

- Field of view (center instrument az, el and delta az, delta el)
- sensitivity
- resolution (usually specified as x mm @ 10m)
- scanning mode (speed vs. range)
- Remove low intensity pixels
- Remove filtered/overloaded pixels

Return/Export Data

The LASSI DAQ program will emit status periodically over the PUB-SUB connection to indicate the state of the scan. At scan completion, TLS data will be published to the PUB connection endpoint.

Scan Metadata

The following scan metadata will be returned when requesting image data:

- orientation matrix – currently unused in processing
- ambient conditions (temp, press) as reported by TLS
- TLS project, scan number (Not to be confused with GBT project and scan number)

3.4.3 Context Diagram

Section 3.3.3 describes the LASSI DAQ context.

3.4.4 Architecture Background

The LASSI DAQ was designed conforming to the following constraints:

Windows/C# Programming Environment

The lassi_daq will require a windows machine to be run on. The driver for this decision is a constraint that the TLS vendor (Leica) imposes by their application programming interface (API) libraries. The API provided supports only the C# language, running on Windows.

TLS to Host Network Configuration

The TLS and the TLS API use a ‘zero-configuration’ mechanism, where the TLS broadcasts ‘hello’ packets, and waits for the host computer to respond. This works well if the computer is directly connected to the TLS, but is incompatible with GBO site LAN systems. Thus the TLS will need to be directly connected to a host, and the host must provide DHCP services to assign an IP address to the TLS when it boots.

Threading Model

The official documentation does not specify limits on the threading environment for the API. Some calls seem to be asynchronous. For example calling StartScan() returns immediately, even though the scan has yet to execute. Status queries can also be made at any time.

Looking at the ScannerAPIDemo code, many API calls are from non-UI threads/Tasks. This may require a query to Leica about thread safety of the API.

We envision a thread will be dedicated to handling the commands received via ZeroMQ. The same thread could also periodically gather the scanner status.

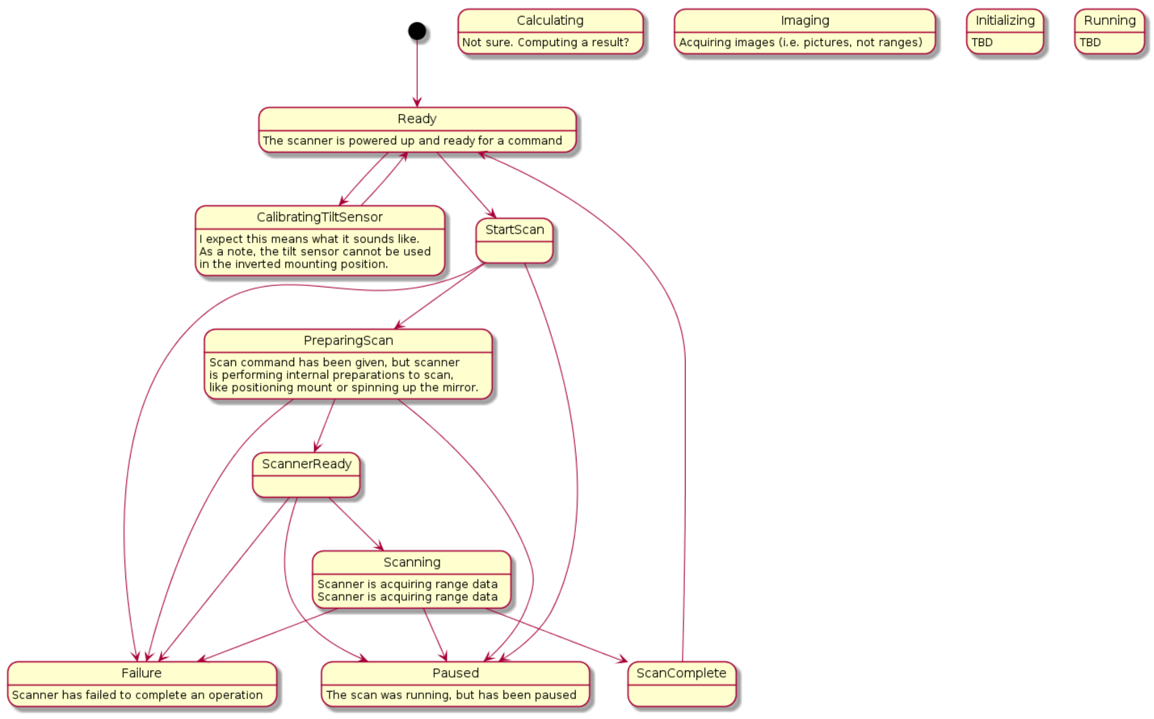
Where inter-thread communications is required, there is a ConcurrentQueue class which functions similarly to the tsemfifo template in Ygor.

3.4.5 Related Views

TBD. This section will be developed during the detailed design phase.

3.5 TLS Scanner State Machine View

3.5.1 Primary Presentation


Calculating
Not sure. Computing a result?
Imaging
Acquiring images (i.e. pictures, not ranges)
Initializing
TBD
Running
TBD
Ready
The scanner is powered up and ready for a command
CalibratingTiltSensor
I expect this means what it sounds like.
As a note, the tilt sensor cannot be used
in the inverted mounting position.
StartScan
PreparingScan
Scan command has been given, but scanner
is performing internal preparations to scan,
like positioning mount or spinning up the mirror.
ScannerReady
Scanning
Scanner is acquiring range data
Scanner is acquiring range data
Failure
Scanner has failed to complete an operation
Paused
The scan was running, but has been paused
ScanComplete

Note that this state diagram is created from observations of the scanner hardware while in use.

3.5.2 Element Catalog

The ScanDeviceInterface::get_state() method can be used to query the instruments state. The possible states are:

- Ready – The scanner is powered up and ready for a command
- Paused – The scan was running, but has been paused
- Calculating – Computing a result
- Imaging – Acquiring images (i.e. pictures, not ranges)
- Calibrating Tilt Sensor – Note that the tilt sensor cannot be used in the inverted mounting position.
- Preparing Scan – Scan command has been given, but scanner is performing internal preparations to scan, like positioning mount or spinning up the mirror.

- Scanning – Scanner is acquiring range data
- Initializing – TBD
- Running – TBD
- Failure – Scanner has failed to complete an operation

3.5.3 Context Diagram

Section 3.3.3 describes the TLS Scanner context.

3.5.4 Architecture Background

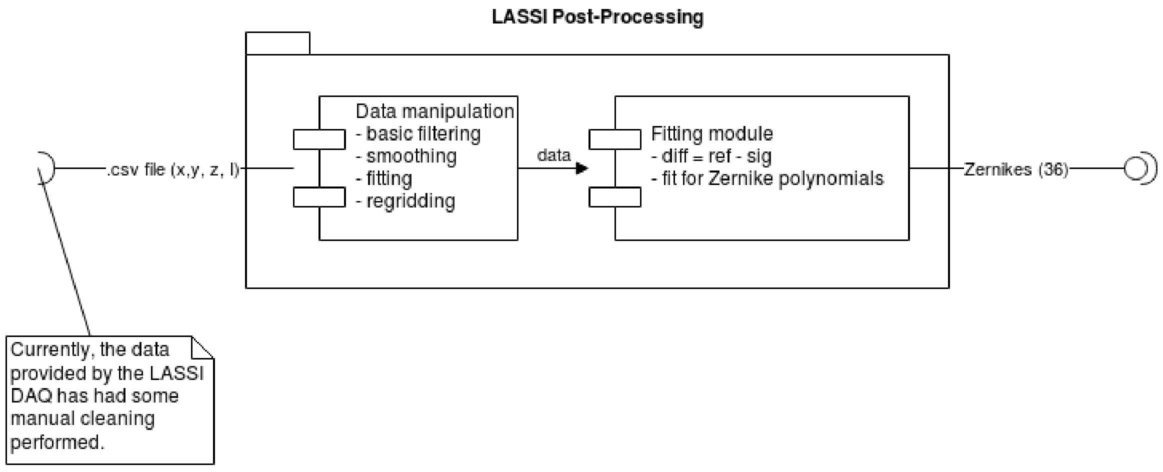
This state machine describes the behavior of the TLS scanner’s firmware. This is a COTS component of LASSI, and is a constraint that the LASSI DAQ is designed against.

3.5.5 Related Views

TBD. This section will be developed during the detailed design phase.

3.6 LASSI Post-Processing Decomposition View

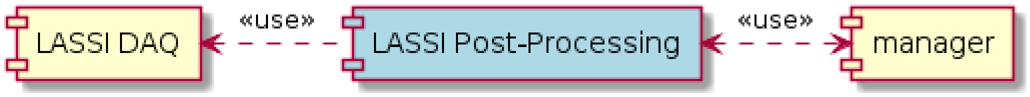
3.6.1 Primary Presentation



3.6.2 Element Catalog

- Data manipulation – module performing various operations to prepare data for fitting
- Fitting – module taking the difference between scans, and fitting this difference to a calculated Zernike polynomial

3.6.3 Context Diagram



3.6.4 Architecture Background

The LASSI Post-Processing component is based on Fred Schwab’s original prototyping, done with Mathematica. The TLS Scanner produces a point cloud, which then must be filtered to include only relevant parts of the scan (i.e. removing any data from the TLS Scanner that is not the surface of the primary reflector of the GBT). The LASSI Post-Processing must then smooth this data and fit a parabola to the result. The difference between the smoothed data and the fit puts the data in the direction of the antenna boresight. Then the data is regridded to be made evenly spaced in the x-y plane. At this point, the data from this TLS measurement is considered 'processed'.

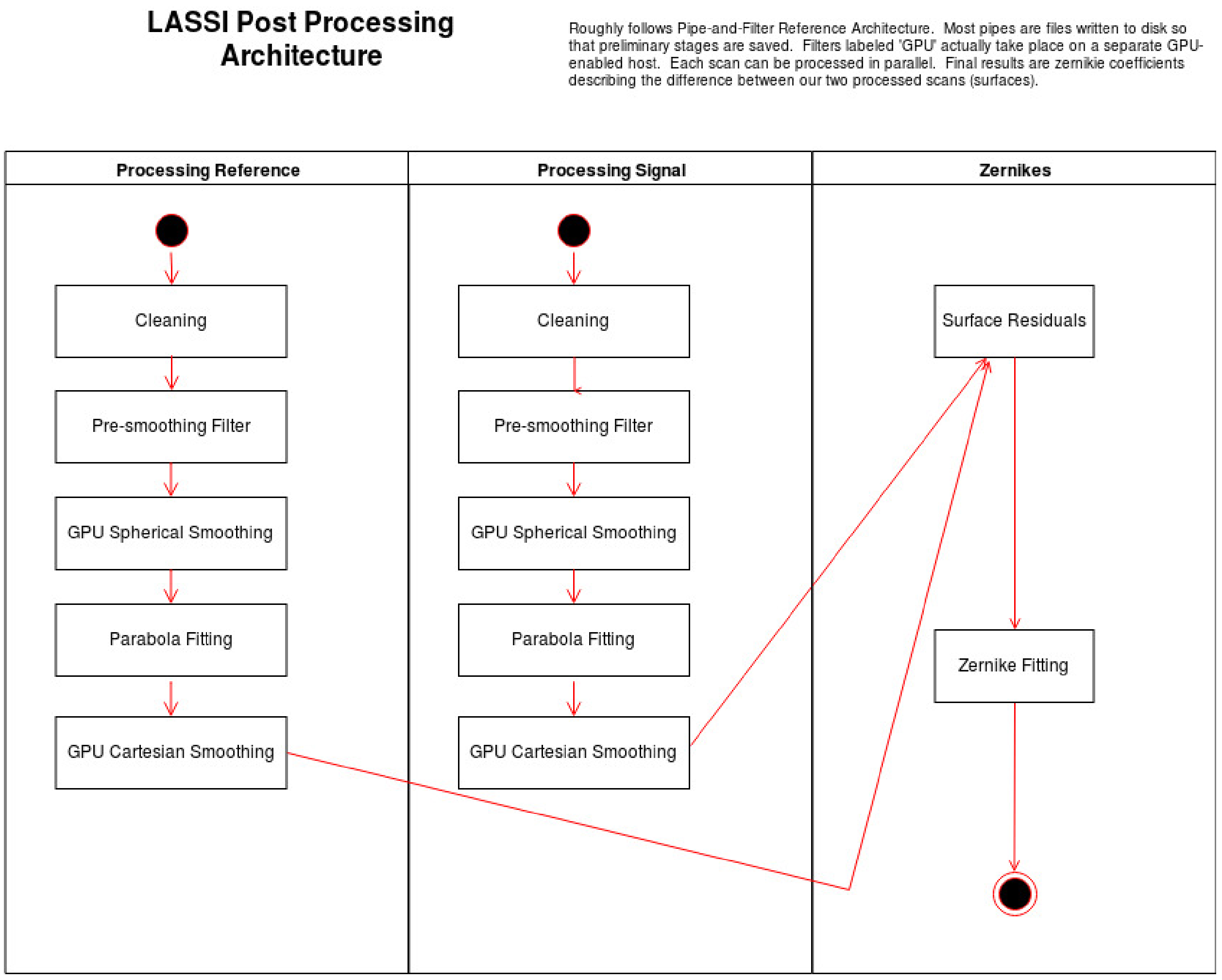
A subsequent TLS measurement is made and is processed identically. With two 'processed' measurements, we can now fit the residuals between the measurements and fit these to Zernike polynomials. These polynomials, in turn, can be used to command the Active Surface to compensate for the deformations seen between the two scans.

3.6.5 Related Views

TBD. This section will be developed during the detailed design phase.

3.7 LASSI Post-Processing Data Flow View

3.7.1 Primary Presentation



Roughly follows Pipe-and-Filter Reference Architecture. Most pipes are files written to disk so that preliminary stages are saved. Filters labeled 'GPU' actually take place on a separate GPU-enabled host. Each scan can be processed in parallel. Final results are zernikie coefficients describing the difference between our two processed scans (surfaces).

3.7.2 Element Catalog

Wallclock time estimates for each element's expected runtime is noted in parenthesis.

- Cleaning and Pre-Smoothing Filters (~1.5 min) – these filters remove all data points that are not of interest, such as the ground, cross-beams, etc. and does everything necessary before we can smooth the data, including rotations. This step starts with 12e6 data points; the processed data set is 8e6 data points.
- GPU Spherical Smoothing (~6 min) – this filter uses GPUs to convert the Cartesian data to spherical coordinates, apply a Gaussian kernel smoothing to create evenly spaced data

in spherical coordinates, and return the data back to Cartesian. This effectively down-samples the data to a 512x512 grid.

- Parabola Fitting (~msec) – this filter fits the smoothed data to a rotated parabola, and finds the difference between this fit and the rotated data, which is the scanner data along the bore site.
- GPU Cartesian Smoothing (~10 sec) – this filter uses GPUs to convert our evenly sampled image of the dish in spherical (the 'guitar pick' image) to an evenly sampled image in Cartesian (the dish as we would expect it). This is currently implemented in Python.
- Surface residuals (~msec) – this filter finds the difference between two processed scans.
- Zernike Fitting (< 1 min)– this filter uses the Opticspy package (http://opticspy.org/) to fit Zernike polynomials to our surface residuals.

3.7.3 Context Diagram

See Section 3.6.3 for LASSI Post-Processing context.

3.7.4 Architecture Background

The post-processing module is the most time-consuming part of LASSI, as well as the one with the fewest external constraints. In this design, we have looked at multiple methods to reduce the post-processing computation time. The following is a list of tactics that could improve performance, prioritized considering both ease of implementation and the expected increase.

1. Binning – Using a binning or “windowing” strategy in the smoothing step may reduce the time needed for these calculations, as well as allow the use of multiple GPUs and parallelization of the calculation.
2. Shared memory – Keeping both the Python and C++ implementations, but use shared memory instead of files to communicate between modules. This might be done using MATRIX or other shared-memory tools.
3. Remove diagnostics and debugging code – Remove diagnostics that are currently being used, but may not be needed for a production system. The current post-processing prototype writes the data to disk between the various stages to facilitate easier debugging and analysis. The finalized version of the module will keep data in memory between each step in the data flow, eliminating this slowdown.
4. Run outside of Jupyter notebooks – The post-processing is currently being run using Jupyter Notebooks to facilitate easy exploring of data and approaches to implementing the needed calculations. Running these steps outside of Jupyter may eliminate some overhead.

5. Streaming from the TLS Scanner – if it were possible to stream data from the TLS scanner hardware, the post-processing module could begin data reduction without having to wait for the entire scan. While not possible with the current API, we are investigating if updates from the vendor (Leica) might give us this capability. Beginning smoothing operations before the scan is complete requires streaming the data from the scanner; otherwise, we must wait for the TLS measurement data to be completed.

6. Stream from DAQ – The LASSI DAQ has the ability to stream the data using ZeroMQ; the post-processing module will read the TLS measurement data from this ZeroMQ stream, eliminating the slowdown that would be caused by writing to disk or accessing the data via NFS.

7. Increase network speed – The network connection between the scanner, the LASSI DAQ, and the post-processing module might be improved. Initial tests were done with a 100 Mbps network connection; moving to a 1 Gbps connection should improve data transfer speeds between the scanner and the DAQ.

8. Use Cython – Implement Python modules using Cython (https://cython.org/).

9. Use C++ – Move all code to C++; this would allow us to easily keep all data in memory as we move between post-processing steps. Currently, the post-processing prototype is being written in Python; as we finalize the design, part or all of the module may see performance improvements by being implemented in C++ or other compiled language.

10. Increase hardware resources – Provide more resources on the host machine.

11. Use PyCUDA – Move all code to Python, using PyCUDA (https://mathema.tician.de/software/pycuda/) to implement the GPU processing; this would allow us to easily keep all data in memory as we move between post-processing steps. This may allow us to more easily leverage the GPU hardware for performance increases.

12. Use vendor-supplied filtering – The TLS scanner and related software from the vendor has some options to do basic filtering, and can use GPUs to accelerate those operations.

13. Use multiple GPUs – The LASSI DAQ may be able to export data faster by changing the API the post-processing module uses. The post-processing module is designed to run on a GPU-enabled host so that we can accelerate calculations using that hardware. By adding multiple GPUs (i.e. increasing the resources available), we may be able to obtain further performance gains.

14. Use lower resolution TLS measurements – Adjusting the resolution of the scanner to decrease measurement time.

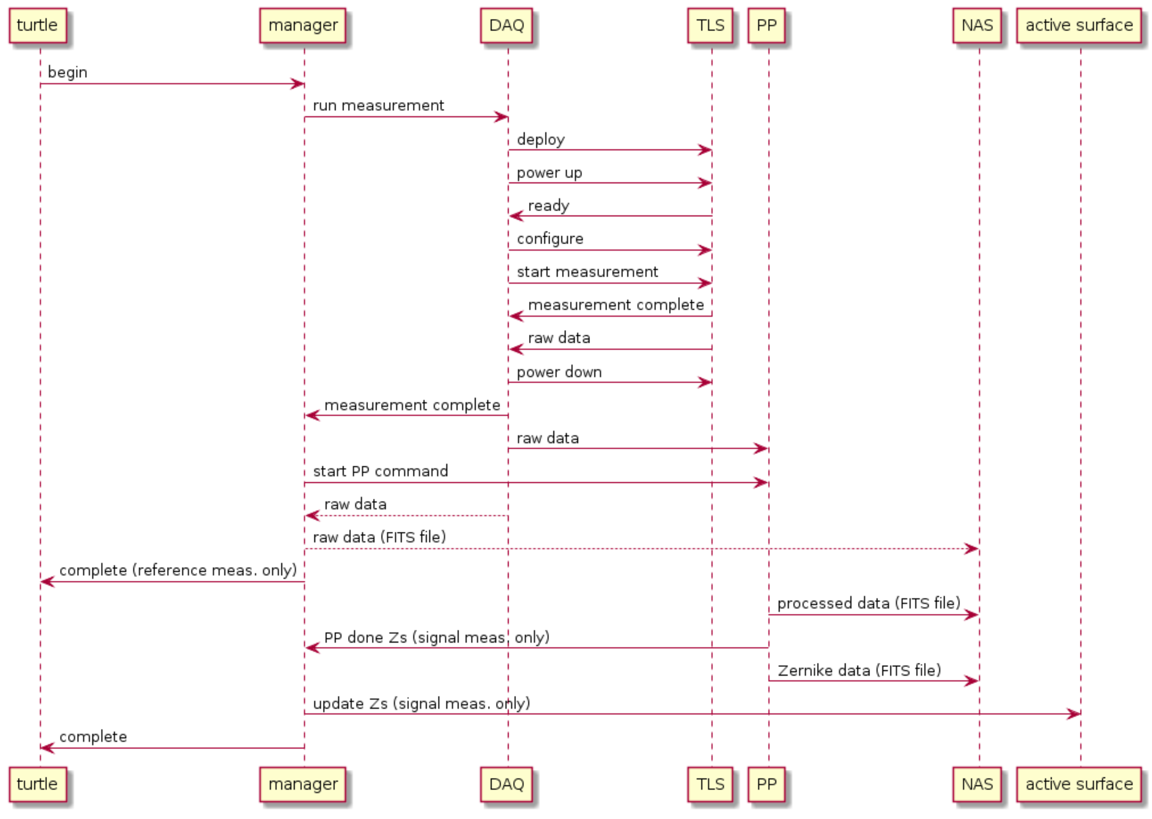
15. Use Dask – It may be possible to implement parallelism in Python using Dask (https://dask.org/) .

3.7.5 Related Views

TBD. This section will be developed during the detailed design phase.

3.8 LASSI Sequence View

3.8.1 Primary Presentation



3.8.2 Element Catalog

- Turtle – the observation management system that loads, edits, and validates Scheduling Blocks, submits them to the telescope for execution immediately or dynamically, and monitors the progress of execution.
- manager – software that acts as a control interface for GBT hardware.
- DAQ – software commanding the TLS scanner and transferring data to and from it. The output of this component will be data from the TLS measurement in (x, y, z, I) format.
- TLS – Leica scanner hardware.
- PP – GBO-developed software that takes in point clouds from the scanner, does any required data reduction, and returns calculated Zernike coefficients that describe the GBT active surface.

- NAS – Network storage system used by various GBT components to store and read data. Both the post-processing and LASSI manager components read and write FITS data to this location using the file system.
- active surface – the software controlling the active surface of the GBT that can adjust the shape of the dish using Zernike polynomials as input.

When processing is requested, to the M&C system the manager will sequence through the standard manager states. A normal sequence might be:

1. Ready to Activating
2. Activating to Running
3. Running to Stopping
4. Stopping to Ready

The manager can indicate a fatal error by entering the Aborting state from any state except Ready.

Manager to Lassi_daq Sequencing

When processing is started (i.e. the Manager receives an activate via its Panel interface) the lassi_daq is sent configuration settings and told to begin a TLS scan. The lassi_daq controls the deploy/retract interface, so the lassi_daq will cause the TLS to power-up and deploy for measurement.

The lassi_daq will periodically check the TLS state to verify its availability. The lassi_daq will also wait for a (TBD) indication that the TLS has deployed and is ready to scan.

When the TLS is ready, the TLS is configured and sent a ‘start’ command. The TLS will sequence through its states of ‘preparing to scan’, ‘scanning’, ‘processing’ and ‘ready’. When lassi_daq detects the end of the scan, it transfers the scan data from the TLS, and publishes the data to both the data pipeline and to the Lassi Manager, which in turns writes the data into a fits file for archival.

The data pipeline is then given some contextual data (e.g. the reference scan information) and given the go-ahead for processing the new data. While the pipeline is processing, the lassi_daq commands the TLS to shutdown cleanly. After a prefined interval (perhaps 15 seconds), the lassi_daq cuts power to the TLS, and commands it to retract for normal observing.

Manager to Pipeline Sequencing

As noted above, the lassi_daq publishes the point cloud data to the pipeline, which is waiting for a ‘go-ahead’ from the LASSI manager. When a successfully TLS scan has been acquired, the LASSI manager provides the pipeline with the reference scan data, or indicates the current scan will be the next reference scan. The ‘go-ahead’ is given and the pipeline begins processing the scan.

Manager to Active Surface Sequencing

Once the data has been successfully processed, turtle will set the active surface manager Zernike parameter, which will convey the correction required to maintain the surface at an optimum position. Only the parameter is updated. The surface will move into position at the beginning of the next observation.

3.8.3 Context Diagram

See Section 3.1.3 for overall LASSI context.

3.8.4 Architecture Background

The expected sequence described in this view was chosen to mirror similar procedures already being used with the GBT, specifically the AutoPeak process. (See http://docs.greenbankobservatory.org/sparrow/). This approach also allows us to store data from the calibration scan using the FITS file format and the GBO network attached storage (NAS), the same locations and file format used to store other data from the GBT. By using existing infrastructure, including deployment hardware and existing software interfaces (e.g. the Manager base class), we take advantage of both in-house expertise and proven designs and components.

In order to best utilize the current control infrastructure of the GBT control system, the LASSI manager will be built. It will be responsible for:

1. receiving commands and parameter settings from the turtle server;
2. handing the states and status of the lassi_daq data acquisition program (which in turn manages the TLS itself);
3. handing the sequencing of TLS scan acquisition and sub-sequence pipeline processing;
4. relaying the pipeline computed results (adjustment Zernike coefficients) to the active surface manager;
5. archiving the raw TLS data into fits files;
6. retaining the necessary state between LASSI TLS scans (e.g. the identity of the previous reference scan, and related data);
7. reporting Manager state transitions, and illiciting M&C error/status messages when necessary;

8. providing subsystem (e.g. lassi_daq, post-processing, etc.) progress as feedback to the turtle server;
9. aborting the processing if either requested by the user/turtle server or if an error is detected

Manager Details

The GBT control system uses programs, called ‘Managers’, which present a consistent interface and sequencing for sub-systems on the GBT. Sets of Managers are controlled via the ScanCoordinator which acts as the scan sequencing and coordination program. The turtle server communicates with the ScanCoordinator to initiate and end scans. A LASSI Manager will handle the TLS subsystems.

Managers contain ‘parameters’ – the data for configuring specific items which control Manager and related subsystem behaviors. These are listed in an appendix (see section 9).

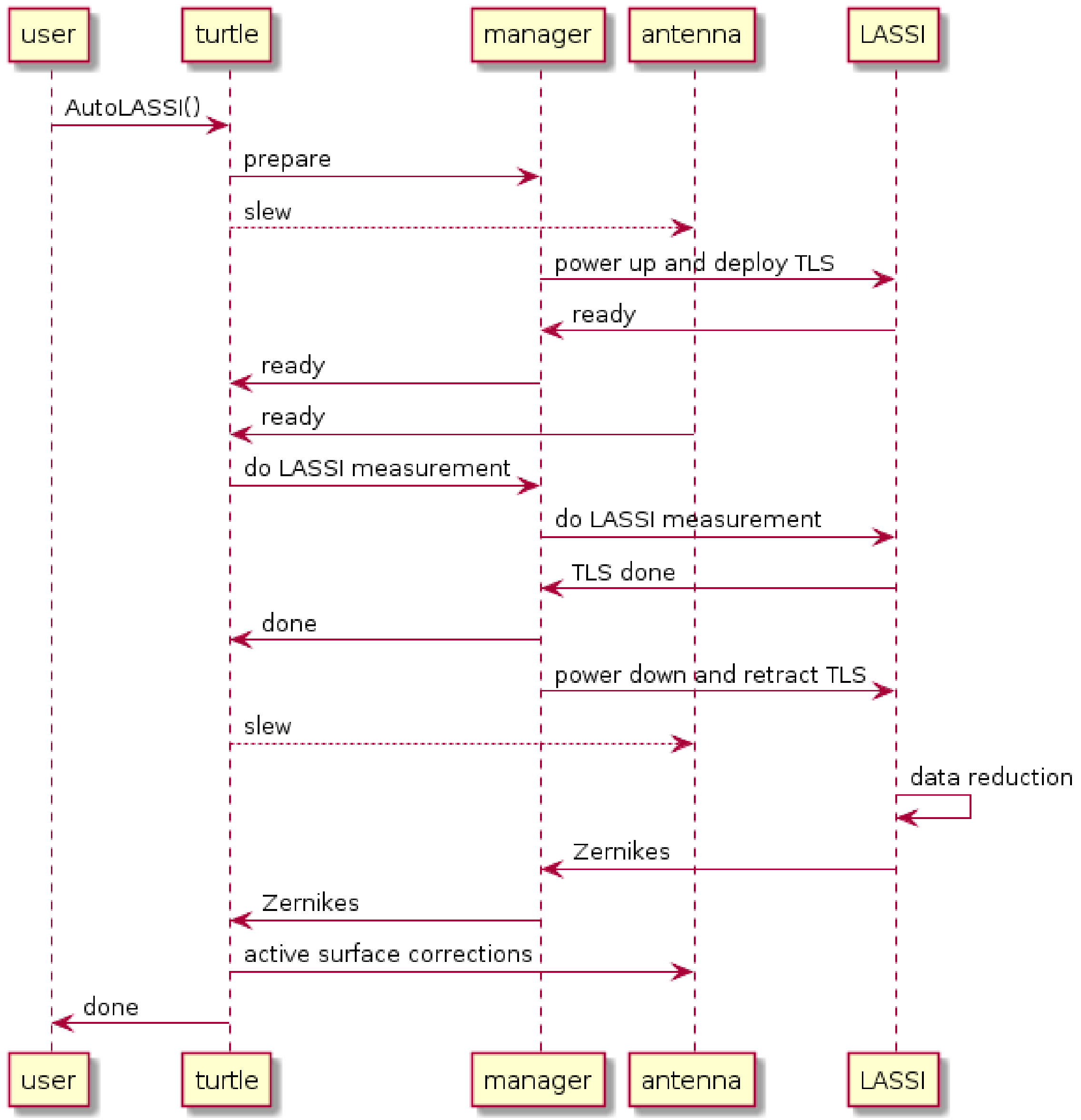
In the case of a reference scan, the LASSI manager should indicate a complete status once the TLS data has been acquired. This will allow observing to continue which the reference scan is being processed. There is no dependence (i.e. the active surface is not updated) on a reference scan. In contrast, when processing a non-reference scan is processed, the manager should not return to ready until the pipeline is finished processing and the active surface has been updated.

3.8.5 Related Views

TBD. This section will be developed during the detailed design phase.

3.9 Interoperability Sequence View

3.9.1 Primary Presentation



3.9.2 Element Catalog

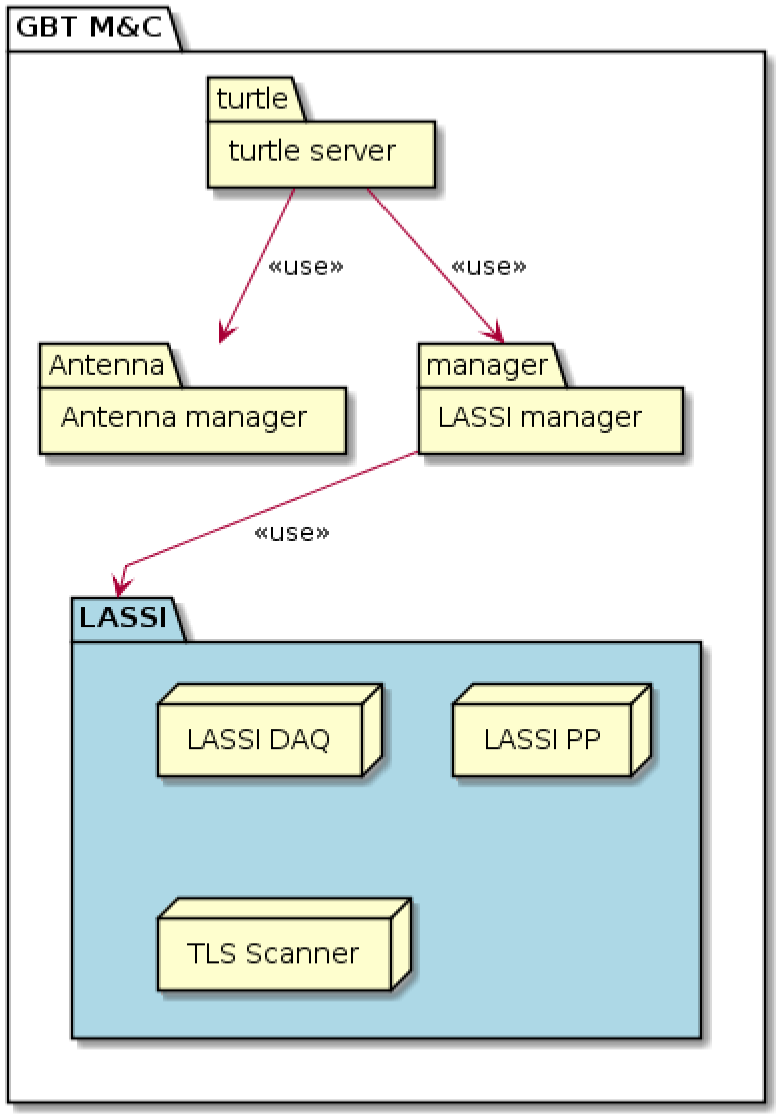
- User – observer, operator, or other user of the GBT

- turtle – the observation management system that loads, edits, and validates Scheduling Blocks, submits them to the telescope for execution immediately or dynamically, and monitors the progress of execution.
- Manager – the LASSI manager; the software abstraction used by the GBT M&C system for monitor and control of LASSI
- antenna – the software managers controlling GBT antenna and active surface.
- LASSI – the LASSI system, including the TLS scanner, the LASSI DAQ, post-processing; in this view, however, it does *not* include the LASSI manager.

Actions

- AutoLASSI() – the command run an AstrID user, requesting a LASSI active surface correction.
- Prepare – a requested state change for the manager; this state sets any parameters as appropriate and does any other needed setup.
- Slew – commanded antenna motion. Moving the antenna to the requested position can take up to ~10 minutes (~20 minutes in cold weather), but is a non-blocking operation i.e. other operations can be performed while the telescope is in motion, including deployment of the TLS scanner.
- Power up and deploy TLS – booting the TLS scanner, deploying it from its housing to ready position, and any associated operations.
- Ready – a system reporting that it has completed requested operations and is waiting for the next command.
- Do LASSI measurement – perform a LASSI measurement, outlined in TBD.
- TLS done – the TLS scanner has completed its measurement; at this point, the remaining work for LASSI is all in data reduction and calculation, and the hardware is no longer required.
- Done – the manager reports completion of the requested state change.
- Data reduction – post-processing, comparison, and other computations needed to calculate the Zernike polynomials for active surface corrections.
- Zernikes – the Zernike polynomials describing corrections to the active surface.
- Active surface corrections – commands to the active surface to modify its shape.

3.9.3 Context Diagram



3.9.4 Architecture Background

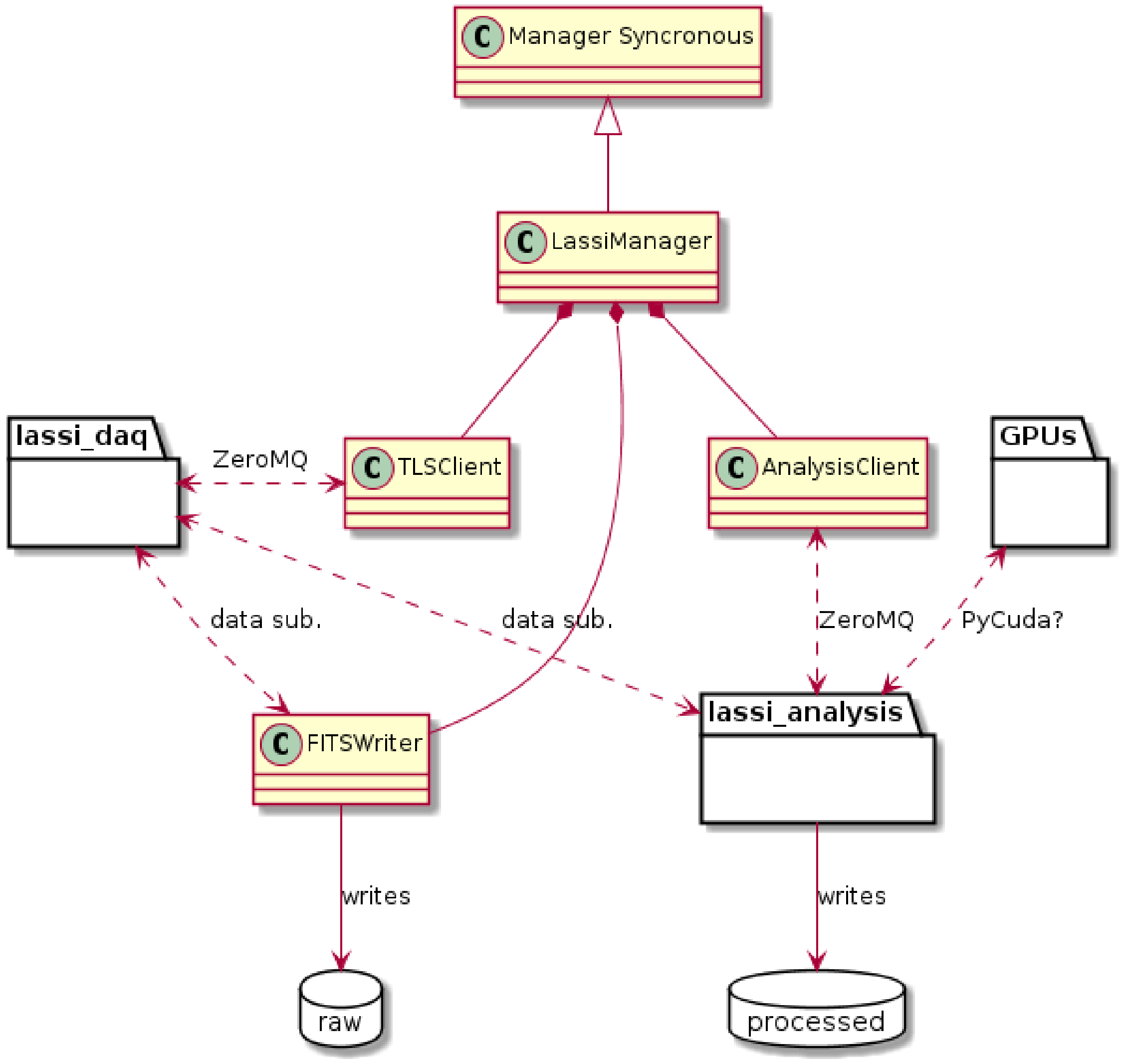
The driving motivation of LASSI is to increase the amount of time available for the GBT to do high-frequency observations. Given that, we want to minimize the time LASSI itself takes to measure and correct the active surface. By being aware of the larger M&C system in which we are operating, we can perform some LASSI operations in parallel with other telescope commands, minimizing the time LASSI block other activities such as telescope slewing.

3.9.5 Related Views

TBD. This section will be developed during the detailed design phase.

3.10 LASSI Manager Class View

3.10.1 Primary Presentation



3.10.2 Element Catalog

- Manager Synchronous – this is an abstract class used as the base for M&C managers that require synchronous communication.
- LassiManager – the specialization of the Manager Synchronous class, used to implement the LASSI manager.

- TLSClient – a helper class for the LassiManager, responsible for communicating directly with the TLS hardware.
- AnalysisClient – a helper class for the LassiManager, used to perform data analysis.
- FITSWriter – a helper class for LassiManager, used to write data to FITS files.
- lassi_daq – software external to the LASSI manager that is used as the interface to the hardware. Communication between the lassi_daq and the TLSClient will be done using ZeroMQ.
- lassi_analysis – software external to the LASSI manager, used to process the data, calculate Zernike coefficients, etc. Communication between lassi_analysis and the AnalysisClient will be done using ZeroMQ. The lassi_analysis component will also subscribe to data provided by lassi_daq through ZeroMQ.
- GPUs – in order improve performance of lassi_analysis, some calculations may be done on GPU hardware. Moving the data to the GPU and back will be done using PyCUDA.
- Raw – this component represents the raw data from the lassi_daq written in FITS format to storage.
- Processed – this component represents the processed data, including the final Zernike corrections written to storage. The format of this data is TBD.

3.10.3 Context Diagram

TBD

3.10.4 Architecture Background

The LASSI manager brings together the other software components of the system, including data acquisition from the TLS scanner itself, data processing routines, and storing both the raw and processed data. Where communication is needed with components outside of the LASSI manager implementation, we use ZeroMQ; this gives us a flexible, non-blocking means of communication between components that has mature implementations in many languages, including C++ and Python.

3.10.5 Related Views

TBD. This section will be developed during the detailed design phase.

3.11 GFM Sequence View

3.11.1 Primary Presentation

3.11.2 Element Catalog

- TBD

3.11.3 Context Diagram

TBD

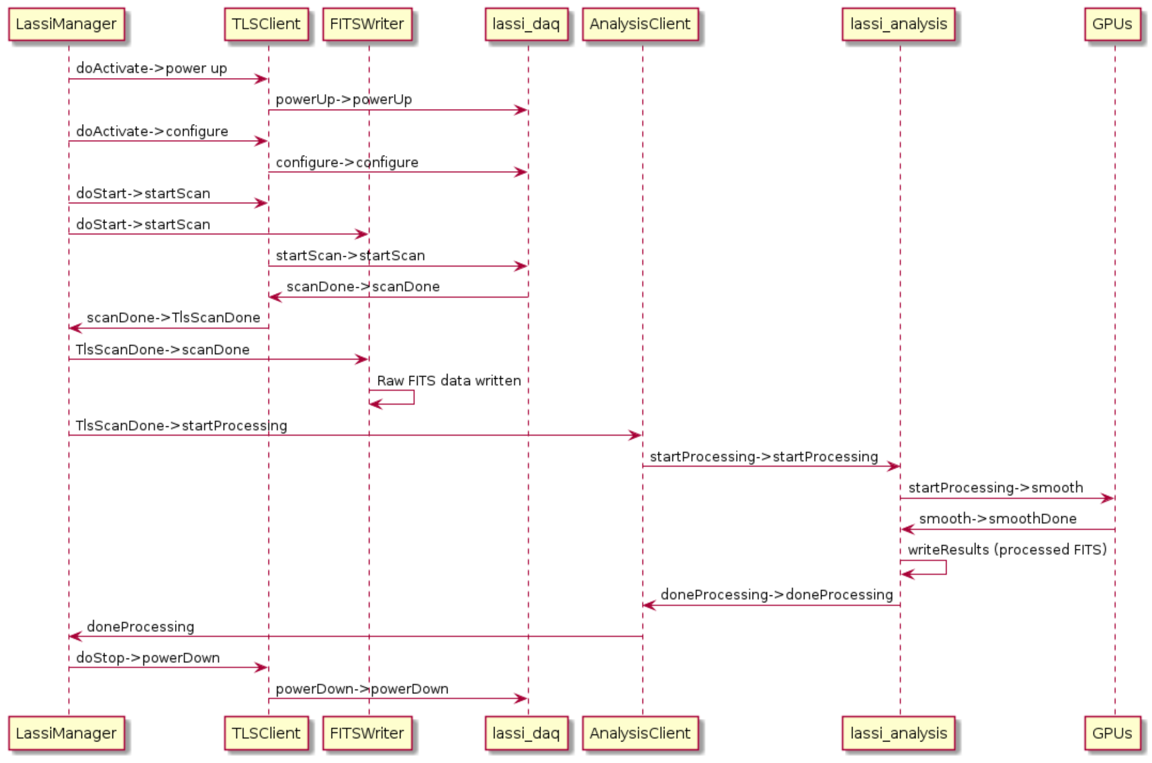
3.11.4 Architecture Background

3.11.5 Related Views

TBD. This section will be developed during the detailed design phase.

3.12 LASSI manager Sequence View

3.12.1 Primary Presentation



3.12.2 Element Catalog

- LassiManager – the LASSI manager implementation
- TLSClient – the helper class to interface with the TLS client
- FITSWriter – the helper class to interface that writes FITS files
- lassi_daq – the component that controls the TLS scanner
- AnalysisClient – the helper class that serves as an interface to lassi_analysis
- lassi_analysis – the component that performs data reduction and analysis
- GPUs – code running on GPUs using CUDA

3.12.3 Context Diagram

TBD

3.12.4 Architecture Background

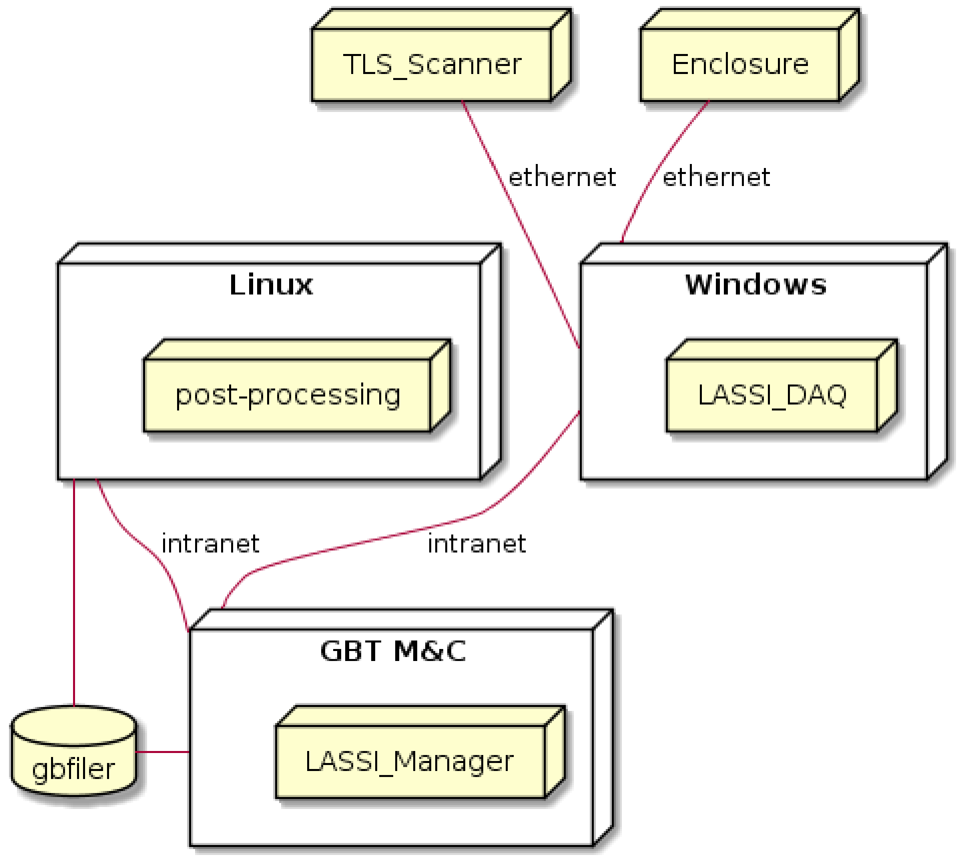
The overall architecture pattern used by LASSI is pipe-and-filter, and this view details the expected sequence of the LASSI manager itself.

3.12.5 Related Views

TBD. This section will be developed during the detailed design phase.

3.13 Deployment View

3.13.1 Primary Presentation



3.13.2 Element Catalog

Elements

- LASSI_DAQ – LASSI data acquisition component
- post-processing – LASSI post-processing software
- TLS_Scanner – Leica hardware scanner
- Enclosure – hardware controlling mounting, weather enclosure, and other mechanical concerns

- Windows – computer running Windows 10; this is needed because the TLS Scanner only provides a .NET interface. This machine may also include GPU hardware, depending on the needs of the LASSI Post-Processing component.
- Linux – computer running RHEL7
- GBT M&C – monitor and control system for GBT. This is a collection of Linux machines, connected by a private network and coordinated by TaskMaster.
- NAS – network storage system used by various GBT components to store and read data. Both the post-processing and LASSI manager components read and write FITS data to this location using the file system.

Relationships

- Ethernet link – physical Ethernet connection using Cat6e or optical fiber with media converters.
- Intranet – TCP/IP network on the GBT. These will be at least 100 Mb connections, and GBO's Computing division does not anticipate any issues with this deployment design.

3.13.3 Context Diagram

The specific hardware allocations are TBD; the context diagram will be added when hosts are chosen for specific modules. This architecture assumes that all components of LASSI are deployed on the GBT or the GBO internal network.

3.13.4 Architecture Background

Where ever possible, LASSI should be deployed on hardware running Red Hat Enterprise Linux; this is what the majority of the GBT M&C system runs on, and thus makes LASSI easier to maintain, modify, and integrate. The notable exception to this is the LASSI DAQ. The API provided by Leica for the TLS scanner is based on .NET, and must be run on Windows.

We explored ways of using the Leica API on Linux; unfortunately, some .DLLs provided by the scanner manufacturer caused illegal instructions when run under Mono. There may also be warranty or support issues from the manufacturer if we are not running on an officially supported platform. Given these issues, we have designed LASSI to be deployed with the DAQ running on a Windows host.

LASSI must be able to go from a commanded measurement to issuing commands to the GBT M&C system within the time constraints determined by the science goals. Since the TLS Scanner has fixed times for scanning, we anticipate the performance bottleneck to the the TLS Post-Processing component. As such, it may be deployed on and accelerated with GPU hardware.

The LASSI manager will be deployed on a host running other, similar managers for the GBT, and will have the ability to communicate with the rest of the M&C system as well as writing FITS files to record data in the same manner as other managers.

3.13.5 Related Views

TBD. This section will be developed during the detailed design phase.

3.14 Work Assignment View

3.14.1 Primary Presentation

Component	**Staff**
LASSI DAQ	Joe Brandt
Post-Processing	Paul Marganian, Andrew Seymour, Pedro Salas
Manager	Paul Marganian, Nathaniel Sizemore
Turtle client/server	Paul Marganian, Nathaniel Sizemore

3.14.2 Element Catalog

N/A

3.14.3 Context Diagram

N/A

3.14.4 Architecture Background

Assignments based on expertise.

3.14.5 Related Views

TBD. This section will be developed during the detailed design phase.

4 Relations Among Views

Each of the views specified in Section 3 provides a different perspective and design handle on a system, and each is valid and useful in its own right. Although the views give different system perspectives, they are not independent. Elements of one view will be related to elements of other views, and we need to reason about these relations. For example, a module in a decomposition view may be manifested as one, part of one, or several components in one of the component-and-connector views, reflecting its runtime alter-ego. In general, mappings between views are many to many. Section 4 describes the relations that exist among the views given in Section 3. As required by ANSI/IEEE 1471-2000, it also describes any known inconsistencies among the views.

4.1 General Relations Among Views

CONTENTS OF THIS SECTION: This section describes the general relationship among the views chosen to represent the architecture. Also in this section, consistency among those views is discussed and any known inconsistencies are identified.

TBD. This section will be developed during the detailed design phase.

4.2 View-to-View Relations

CONTENTS OF THIS SECTION: For each set of views related to each other, this section shows how the elements in one view are related to elements in another.

TBD. This section will be developed during the detailed design phase.

4.3 Use Case Mapping

The following table lists the use cases as described in the Architecture Plan, and describes the components of the LASSI architecture that handle each case.

Use Case	**Views**	**Architecture solution and notes**
Execute AutoOOF	N/A	The LASSI system is not responsible for the initial AutoOOF scans; that data and the initial corrections to the active surface are commanded by AstrID.
Conduct initial TLS measurement	3.3, 3.6, 3.13	The initial TLS measurement exercises the entire LASSI system, beginning with

Use Case	**Views**	**Architecture solution and notes**
		AstrID requesting the measurement be taken and ending with processed data that can be used to calculate Zernike polyno-mials.
Store TLS results	TBD	Storage between the initial scan and the difference has not been decided.
Compare TLS measurements	3.6	Calculating the differences between the scans is done by the LASSI post-process-ing module, and results in a calculated Zernike polynomial that can be given to the GBT M&C active surface manager.
Change shape of prime reflector	3.6	After LASSI has calculated the Zernike polynomial needed to return the active surface to its desired shape, it provides that data to the GBT M&C system, which is responsible for incorporating that data with other corrections (gravity, thermal variations, etc.) and moving the panels as needed.

5 Referenced Materials

Architecture Plan	Whitehead, *Enhancing GBT Metrology to Support High Resolution 3mm Molecular Imaging for the U.S. Community Architecture Plan,* <https://sharepoint.nrao.edu/pmd/projects/613%20GBT%20Metrology/Software/Architecture%20Plan/613%20GBT%20Metrology%20Architecture%20Plan%20v01.docx?Web=1>
Barbacci 2003	Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs)*, Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html>.
Bass 2003	Bass, Clements, Kazman, *Software Architecture in Practice,* second edition, Addison Wesley Longman, 2003.
Clements 2001	Clements, Kazman, Klein, *Evaluating Software Architectures: Methods and Case Studies,* Addison Wesley Longman, 2001.
Clements 2002	Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford, *Documenting Software Architectures: Views and Beyond*, Addison Wesley Longman, 2002.
ConOps	Bloss et al, *Enhancing GBT Metrology to Support High Resolution 3mm Molecular Imaging for the U.S. Community Concept of Operations,* <https://sharepoint.nrao.edu/pmd/projects/613%20GBT%20Metrology/Systems%20Engineering/Concept%20of%20Operations/613%20GBT%20Metrology%20Concept%20Documents%20v001_3%20181009.docx?Web=1>

6 Directory

6.1 Glossary of Terms

The following specialized terms are used in the LASSI project.

Term	Definition
AOCS	AutoOOF Corrected Surface – this describes the GBT primary surface after it has been calibrated by AutoOOF. AOCS is assumed to be the “accepted surface”, to which LASSI will return the telescope.
AutoOOF	Software procedure that corrects the shape of the GBT active surface using out-of-focus holography (OOF).
Calibration scan	An observation that uses the GBT’s Scan Coordinator, but does not record data from an astronomical source. Calibration scans are used to exercise the antenna, receivers, etc. and record information, but do not observe anything “on the sky”.
GBT observation	The process of obtaining astronomical information using the GBT, or the data generated from that process. (To avoid confusion, please use this in place of "scan", which is ambiguous.)
LASSI DAQ	Software to obtain data from the TLS scanner and provide it in a useful form to other parts of the LASSI system.
LASSI Post-Processing	Software to take data from a TLS measurement and perform data reduction needed by other parts of the system, including the GBT M&C.
Leica	Leica Geosystems, the vendor of the TLS scanner; see https://leica-geosystems.com/
Scan Coordinator	GBT manager that marshals other managers in the M&C system, and ensures that commands are issued in the correct sequence.
software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the

	relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
TLS	Terrestrial Laser Scanner i.e. the Leica device used in the LASSI project.
TLS measurement	A data set about the shape of the GBT's primary or secondary reflector, generated by the Leica laser scanner. (To avoid confusion, please use this in place of "scan", which is ambiguous.)
view	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
Ygor	Base system for the GBT M&C.
Zernike polynomial	A means of describing the shape of a parabola or lens with an n-th order polynomial. For more information, see https://en.wikipedia.org/wiki/Zernike_polynomials

6.2 Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
COTS	Commercial off-the-shelf
DAQ	Data Acquisition
FOV	Field of View
LASSI	Laser Antenna Surface Scanning Instrument
M&C	Monitor and Control
SAD	Software Architecture Document
TBD	To be determined
UML	Unified Modeling Language

7 Appendix - Design History

This section documents design questions that arose during development of LASSI, as well as the decisions made and other options considered. Not all of these issues impact the software architecture directly, but are provided here for reference and context.

Why are we calculating Zernike polynomials to model the antenna surface? Wouldn't it be easier to use the data from the scanner to determine the position of the panels, and adjust accordingly?

The ultimate goal of LASSI is to measure the actual shape of the GBT surface and command the M&C system to return it to the desired shape. One solution would be to measure the location of each panel corner, calculate the difference from the desired location, and move the panel by that amount. Unfortunately, this approach relies on extremely precise measurements of individual points on the surface, something that is difficult to achieve in practice.

Instead of attempting to measure individual points, LASSI measures the entire surface, generating a 'point cloud' of several million measurements all over the surface of the GBT. (Note that due to the design and placement of the scanner, these points are not regularly distributed.) While an individual point may not give us the desired precision, the average over a large number of these data points does yield an accurate description of the surface. Using the moving average of the LASSI point cloud to calculate the Zernike polynomial coefficients essentially acts a as an additional low-pass filter, removing noise from the data set and providing a precise calculation of the dish shape.

Additionally, the GBT M&C system can accept Zernike coefficients as inputs when commanded to change the shape of the primary surface, making this step of the LASSI measurement/correction cycle simpler.

Will LASSI measure the GBT’s subreflector? What would be the benefits of doing so?

Measuring the GBT subreflector is not in scope for phase 1 of LASSI.

What is the "point cloud" that Leica refers to? Where is this data stored?

Leica documentation, engineers, and other resources may refer to the "point cloud", which is the data generated by the scanner -- the points it measures. These are stored in the .ptx files generated by the instrument. The .ptx files are flat (plain text) files.

What is the "noise budget", and what impacts this value?

The terms "noise budget" and "error budget" both refer to the active surface RMS (σ). This value is the amount of deviation in the active surface -- how far it may be from an ideal parabolic shape, beyond which the GBT cannot perform desired high-frequency observations. Many values impact this final number, including (but not limited to) precision and accuracy of the actuators, gravitational and thermal corrections, and the LASSI measurements themselves. Currently, the project aims to keep this error budget within 300 microns.

How long does it take the Leica scanner to measure the primary reflector?

The scan time is 3.09 +- 0.04 minutes (after the scanner has "warmed" up).

What is the current surface TLS measurement accuracy (resolution) in um rms?

With the TLS we can detect Z5 (GBT active surface convention) down to 15 microns. If the surface rms is larger than 300 microns it would have a significant effect on high frequency (100 GHz) observations. A "typical" deformation on the surface of the GBT has an amplitude of 100 microns (you might have a couple of these deformations on top of each other). The GBO science staff estimates that we will need a resolution of ~100 microns to produce useful corrections for high frequency observations.

How is the scanner enclosure commanded?

The current design of the enclosure opens when power is applied and closes when power is removed. This design has the closing done via a spring return, so there will be no power to close requirement. There is no command requirement for either open or close, other than the power signal.

Are there any concerns with operating the TLS device inverted (upside-down)?

The scanner includes feature that keeps the scanner from operating when leveling is turned on. We disable leveling (because the level doesn't work upside down) when we use it in the inverted position. We have confirmed with the vendor that the device can be operated in this manner, as the concern is that operating the device at an angle would cause uneven wear on the bearings in the device. This level disable feature is not exposed to the API, and cannot be controlled remotely.

8 Appendix - LASSI DAQ Command Protocol Details

The TLS controller will listen for command at the local port 55666. A URL would look like 'tcp://lassi.ad.nrao.edu:55666'. The command/response link are ZMQ_REQ (request) at the client end and a ZMQ_RPY (reply) at the lassi_daq (TLS scanner controller) end.

Commands and data follow the ZeroMQ convention of a message header followed by 0 or more data frames. ZeroMQ handles the details of grouping message boundaries. The headers are always a Mesgpack encoded string, which in some cases is enough to convey the message intent (e.g a start message header is simply 'StartScan', with no following data). Commands are always acknowledged with a Mesgpack encoded OK_ERROR data structure which conveys several items such as error code, error message string, and device status.

The lassi_daq also has a ZMQ_PUB (publish) interface which the scanner status and data are published on. A client must use a ZMQ_SUB socket to subscribe to the published data. The URL would be 'tcp://lassi.ad.nrao.edu:55668'. In addition to data, state/status changes are also published asynchronously (meaning without a query command) on the publish interface.

8.1 TLS Scanner Command Protocol

The following commands are used to initiate/configure the TLS controller. Where items are quoted, the data is UTF-8 encoded strings. Where the word 'struct' occurs, the data is encoded using msgPack. The ASCII drawn boxes shown in each command section denote a ZeroMQ frame boundary. Data types are described in the following section.

8.1.1 Start Scan

```
+-----------------+
| "StartScan"     |
+-----------------+
```

The reply will be a frame of:

```
+-----------------+
| struct OK_ERROR |
+-----------------+
| struct ScanInfo |
+-----------------+
```

8.1.2 Stop Scan

```
+-----------------+
| "StopScan"      |
+-----------------+
```

The reply will be a frame of:

```
+-----------------+
  | struct OK_ERROR |
  +-----------------+
```

8.1.3 Pause Scan

```
  +-----------------+
  | "PauseScan"     |
  +-----------------+
```

* The reply will be a frame of:

```
  +-----------------+
  | OK_ERROR        |
  +-----------------+
  | struct OK_ERROR |
  +-----------------+
```

8.1.4 Resume Scan

```
  +-----------------+
  | "ResumeScan"    |
  +-----------------+
```

The reply will be a frame of:

```
  +-----------------+
  | struct OK_ERROR |
  +-----------------+
```

Note: Only scans which have been 'paused' can be resumed. A scan which has been stopped cannot be resumed.

8.1.5 Export Scan

```
  +-----------------+
  | "Export"        |
  +-----------------+
```

The reply will be a frame of:

```
  +-----------------+
  | struct OK_ERROR |
  +-----------------+
```

This will cause the last scan taken to be exported to the pub/sub interface.

8.1.6 Configure Scan

This updates the settings for:

- scanner project
- resolution (enum: 500mm@100m, 250mm@100m, 125mm@100m, 63mm@100m, 31mm@100m, 16mm@100m, 8mm@100m)
- sensitivity (enum: Normal, High)

- scan mode (enum: Speed, Range, Medium Range, Long Range). Note that not all are currently valid; Speed, Range are known to work. The remaining scan modes are being investigated.
- Field of view (az_center, el_center, az width, el width)

The above info is packed into a message pack encoded struct ConfScan.

```
+-----------------+
| "Configure"     |
+-----------------+
| struct ConfScan |
+-----------------+
```

The reply will be a frame of:

```
+-----------------+
| struct OK_ERROR |
+-----------------+
```

8.1.7 Move to Azimuth

```
+-----------------+
| "MoveAz"        |
+-----------------+
| struct MoveToAz |
+-----------------+
```

* The reply will be a frame of:

```
+-----------------+
| struct OK_ERROR |
+-----------------+
```

8.1.8 Get Status

```
+-----------------+
| GETSTATUS       |
+-----------------+
```

* The reply will be a frame of:

```
+-----------------+
| struct OK_ERROR |
+-----------------+
```

8.1.9 Export Previous Scan From Scanner

This should export a scan from an arbitrary project. Some scan info may not be available however.

The ScanFrom struct contains:

- The scanner project name

- The scanner scan number

```
+------------------+
| "ExportFrom"     |
+------------------+
| struct ScanFrom |
+------------------+
```

* The reply will be a frame of:

```
+------------------+
| struct OK_ERROR |
+------------------+
```

8.1.10 Delete Scan on the TLS device

This will delete a scan stored on the device.

```
+------------------+
| "DeleteFrom"     |
+------------------+
| struct ScanFrom |
+------------------+
```

* The reply will be a frame of:

```
+------------------+
| struct OK_ERROR |
+------------------+
```

8.1.11 Get Result (pyTLS local method)

This method allows subscribed data to be accessed without a callback.

8.1.12 Data Published on the ZMQ_PUB interface

When a 'EXPORT' or 'EXPORTFROM' command is received, two elements are published. First a frame labeled 'HEADER' is sent which has all the scan configuration information. This is immediately followed by a frame labeled 'DATA'. These are intended for use by the fits writer daemon.

```
+------------------+
| "HEADER"         |
+------------------+
| struct FitsHead |
+------------------+
+------------------+
| "DATA"           |
+------------------+
| struct FitsData |
+------------------+
```

On each TL scanner state change, the new state is published.

```
+------------------+
| STATUS           |
+------------------+
+------------------+
| struct OK_ERROR |
+------------------+
```

* This will have the scanner state info.

* It may also contain any error messages present on the device.

An application may also subscribe to one or more of the data columns. When a scan is exported, the union of all subscribed data columns is published. The arrays are custom encoded to support msgpack_numpy which decodes directly into a NumPy array for speed. Note: This encoding may/will suffer from incompatible endianness. If that is the case, there is a NumPy method which can byte-swap an entire array.

The possible arrays are (listed by message headers):

- X_ARRAY – the X component of data acquired from az,el,r measurements
- Y_ARRAY – the Y component of data acquired from az,el,r measurements
- Z_ARRAY – the Z component of data acquired from az,el,r measurements
- AZ_ARRAY - instrument relative raw horizontal angles
- EL_ARRAY - instrument relative raw elevation angles
- R_ARRAY - radial distance measurements
- I_ARRAY - pixel intensity

8.1.13 Subscribing to Data

The subscribe method takes the name, or list of names of the arrays of interest. It also includes an optional argument of a callback, which should have a signature of string data, numpy_array. e.g:

```
def mycallback(name, numpy_array):
    pass

subscribe("X_ARRAY", mycallback)
```

Note that the mycallback entry may be None. In that case, the subscribed data can be accessed using the get_result(name) method.

8.2 Subscription Details

When a subscription request is processed a number of steps happen:

1. The name is validated
2. If provided, the callback is checked for the correct signature
3. If the callback subtask has not been started it is started.
4. A REQ/RPY connection is made from the call context to the callback subtask to announce the subscription. In response, the subtask calls the ZeroMQ setsockopt() with the subscription filter. This sets up the ZeroMQ subscription filter.
5. The last step is to send a subscribe_array request to the lassi_daq, to announce that the connection is interested in the subscribed data.

The reason for the dual sets of filtering is to minimize publishing traffic. One could argue that the step 4 is unnecessary, since step 5 prevents uninteresting data to be published in the first place.

8.2.1 Protocol Data Structures

```
struct ScanInfo
{
    int scan_number;
};
```

Used by StartScan. Note that the scan_number, and similar metadata, will be assigned and determined by the LASSI manager. In production, the scan number will be assigned passed to the DAQ from the M&C infrastructure; the DAQ will not track this independently.

```
struct OK_ERROR
{
    string error_message;
    int state;
    bool is_ok;
};
```

Used in all command responses to acknowledge whether or not the command succeeded, any detailed error messages, and the current scanner state.

```
struct MoveToAz
{
    float move_to_az;
};
```

Used by the Move_Az command.

```
struct ConfScan
{
    string project;
    int scan_resolution;
    int scan_quality;
    int scan_sensitivity;
    int scan_mode;

    float center_az;
    float center_el;
    float fov_az;
    float fov_el;
};
```

Used by the Configure command.

```
struct FitsHeader;
struct FitsData;

struct ScanFrom
{
    string project;
    int scan_number;
}
```

Used by the ExportFrom and DeleteFrom commands.

9 Appendix - LASSI Manager Parameters

The LASSI manager will be developed for LASSI during the detailed design phase, but is anticipated to be similar in design, functionality, and use as other managers already in use on the GBT. It will also provide a graphical user interface (GUI) to display and change parameters.

The parameters and functionality we anticipate for the manager are outlined here.

Manager Base Parameters

Not listed here are a set of 16 basic parameters which are common to all managers.

Reference Scan Identity

The post-processing module will use a reference scan data to compute changes required to the surface. The manager will retain the identifiers necessary to convey the reference scan identity to the pipeline, or indicate that the scan being requested is an actual reference scan.

TLS Acquisition Configuration

The basic controls for the TLS device are resolution, optimize for range vs. speed, sensitivity, field of view (FOV), etc.

Processing Status/Progress

As processing proceeds, a feedback parameter will update to provide items upstream (i.e. turtle server and user) on the progress of the processing.

Interfaces

The manager will interface with the GBT observing systems such as the turtle server via the standard M&C PanelRemote interface. Other interfaces are:

1. lassi_daq, via the protocol described in this document.
2. The data analysis pipeline – TBD: interface will be developed during the detailed design phase.
3. The active surface manager, using a standard PanelRemote interface.